Hierarchical Galois Key Management Systems for Privacy Preserving AIaaS with Homomorphic Encryption

Joon-Woo Lee
Dept. of Electrical and Computer Engineering, INMC,
Seoul National University
Republic of Korea
joonwoo42@snu.ac.kr

Eunsang Lee*
Dept. of Electrical and Computer Engineering, INMC,
Seoul National University
Republic of Korea
shaeunsang@snu.ac.kr

Young-Sik Kim*
Dept. of Information and Communication Engineering,
Chosun University
Republic of Korea
iamyskim@chosun.ac.kr

Jong-Seon No
Dept. of Electrical and Computer Engineering, INMC,
Seoul National University
Republic of Korea
jsno@snu.ac.kr

ABSTRACT

In the artificial intelligence as a service (AIaaS) system in the client-server model, where the clients provide the data on the cloud and the server processes the data by using the deep neural network in the cloud, data privacy via homomorphic encryption is getting more important. Brakerski/Fan-Vercauteran (BFV) and Cheon-Kim-Kim-Song (CKKS) schemes are two representative homomorphic encryption schemes which support various arithmetic operations for encrypted data in the single-instruction multiple-data (SIMD) manner. As the homomorphic operations in these schemes are performed component-wisely for encrypted message vectors, the rotation operations for various cyclic shifts of the encrypted message vector are required for useful advanced operations such as bootstrapping, matrix multiplication, and convolution in convolutional neural networks. Since the rotation operation requires different Galois keys for different cyclic shifts, the servers using the conventional BFV and CKKS schemes should ask the clients having their secret keys to generate and send all of the required Galois keys. In particular, in the advanced services that require rotation operations for many cyclic shifts such as deep convolutional neural networks, the total Galois key size can be hundreds of gigabytes. It imposes substantial burdens on the clients in the computation and communication cost aspects. In this paper, we propose a new concept of *hierarchical Galois key generation method* for homomorphic encryption 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. Since the server can generate the required Galois keys by using the received small set of Galois keys from the client, the server does not need to request additional Galois keys to the clients or to store all possible Galois keys for future use. For example, if we implement the standard ResNet-20 network for the CIFAR-10 dataset and the ResNet-18 network for the ImageNet dataset 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 we use the proposed three-level hierarchical Galois key system, 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.

CCS CONCEPTS

• **Security and privacy → Key management**; **Public key encryption**.

KEYWORDS

Hierarchical Galois key, Homomorphic encryption, Privacy-preserving machine learning, Public key management, Brakerski/Fan-Vercauteran (BFV) schemes, Cheon-Kim-Kim-Song (CKKS) schemes

ACM Reference Format:
Joon-Woo Lee, Eunsang Lee, Young-Sik Kim, and Jong-Seon No. 2022. Hierarchical Galois Key Management Systems for Privacy Preserving AIaaS with Homomorphic Encryption. In *Proceedings of Conference.* ACM, 20 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Fully homomorphic encryption (FHE) is an encryption scheme which supports the evaluation of arbitrary boolean or arithmetic operations on encrypted data. It is a primary solution for the privacy issue of outsourcing computation, which enables the clients to securely entrust enterprises to processing their private information while preserving privacy. The main application includes machine learning [14, 23, 24], genomic analysis [6, 20, 21], and cloud services [22], and AI-as-a-service (AIaaS) [28]. Especially, the privacy-preserving AIaaS system is deemed to be one of the most promising techniques, where the clients provide the encrypted data on the cloud and the server processes the data by using the deep

*Co-corresponding authors

Conference, 2022,
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

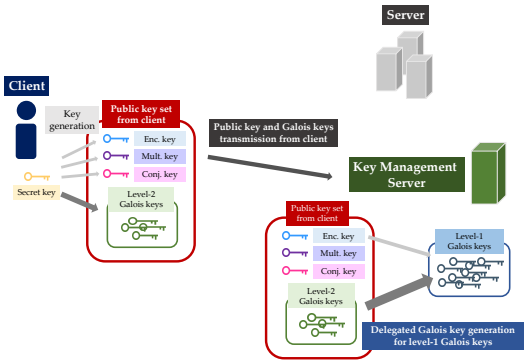
neural network, while preserving privacy of clients' data. Thus, data privacy via homomorphic encryption is getting more important.

Among various FHE schemes, Brakerski/Fan-Vercauteran (BFV) [5, 13] and Cheon-Kim-Kim-Song (CKKS) [9, 11] 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.

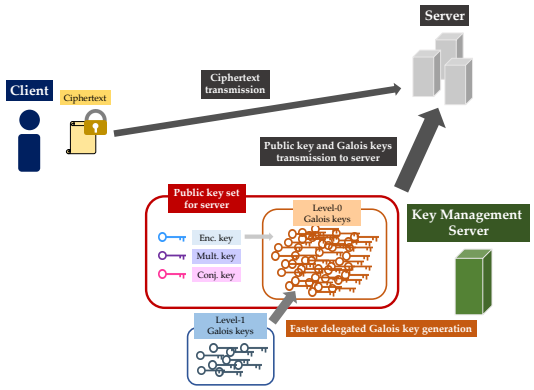
The BFV and CKKS schemes also support rotation operation which corresponds to a cyclic shift of message data within ciphertext. Many applications that require important operations such as bootstrapping, matrix multiplication, and convolution in convolutional neural networks can be achieved using this 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 we should operate data with different positions in one ciphertext, such as the bootstrapping [3, 7, 8, 25, 26], the matrix multiplication [17], and the convolution in convolutional neural networks [19, 23, 24]. 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 we implement the standard ResNet-20 network for the CIFAR-10 dataset with pre-trained parameters with the CKKS scheme with the polynomial modulus degree $N = 2^{16}$ using the techniques in [23], the server requires 265 Galois keys, which occupies 105.6GB of memory in the server. If we design the ResNet-18 network for the ImageNet dataset 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.



(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 1: Efficient Galois key management in three-level hierarchical Galois key generation.

1.1 Our Contributions

In the BFV and CKKS schemes, we observe that the Galois keys can be generated from other Galois keys using key-switching operation. The crucial observation is that we can regard the Galois key as a set of ciphertexts. If we perform the key-switching operation 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, we propose a hierarchical Galois key generation system, 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 this Galois key generation, we also propose several optimization techniques, such as the hoisted Galois key generation and the reduction to graph-theoretic algorithms.

We 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, we also have to prepare the inert client to become an active client at any time. In Figure 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. Our 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.

We conduct the simulation with the proposed Galois key generation system for ResNet models with an appropriate computing environment for the client-server model. If we use a three-level hierarchical Galois key system, 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.

1.2 Outline

Section 2 formalizes the concept of the hierarchical Galois key system and its application to the specific Galois key management. Section 3 deals with the proposed hierarchical Galois key generation algorithm for BFV and CKKS schemes. Section 4 proposes algorithms for efficiently generating a set of Galois keys with the given set of Galois keys in the higher levels. Section 5 suggests a concrete example of the Galois key management protocol and shows the numerical simulation results with ResNet models. Section 6 concludes the paper and suggests future work. Appendices deal with the preliminaries, the proofs of the theorems, and the required cyclic shifts for ResNet models.

2 HIERARCHICAL GALOIS KEY SYSTEM

In this section, we provide an overview of the proposed hierarchical Galois key system. Specific procedures in this system will be described in Sections 3 and 4.

2.1 Definition of Hierarchical Galois Key System

We define the hierarchical Galois key system 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 InitGalKeyGen and GalKeyGen. The algorithm 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 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. We now assume 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 paper 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 3. 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 InitGalKeyGen, and the number of key levels in the hierarchy is one.

2.2 Galois Key Generation Protocol in Hierarchical Galois Key System

We propose a detailed Galois key generation protocol 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 we have, the smaller computation amount we need 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, we need to finely manage this trade-off for the Galois keys.

We assume that we do not know when and what model the client will request the service to the server after the key generation and transmission. We define the *offline phase* 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 1.

Algorithm 1: 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 ℓ 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 ℓ 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) $\{gk_i^{(0)}\}_{i\in\mathcal{T}_0} \leftarrow \mathsf{GalKeyGen}(0, \mathcal{U}_\ell, \{gk_i^{(1)}\}_{i\in\mathcal{T}_\ell}, pk, \mathcal{T}_0)$

3 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, we will deal with them at once.

We use the term *ciphertext* 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 $Q = \prod_{i=0}^{\mathsf{dnum}-1} Q_i$ 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* $\mathsf{gk}_r = \{\mathsf{gk}_{r,i}\}_{i=0,\cdots,\mathsf{dnum}-1}$ 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-switching operation to the ciphertext in the modulus q, where q is a divisor of Q. We call Q 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, we first perform an operation of $(b(X), a(X)) \mapsto (b(X^{5^r}), a(X^{5^r}))$. 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 $s(X^{5^r})$. We have 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.

3.1 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, we set the larger modulus for the Galois key 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. We define hdnum_ℓ 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 hdnum_ℓ 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. We also propose an algorithm for obtaining the list of special modulus for the Galois key in each key level in Algorithms 2 and 3.

Given RNS moduli for ciphertexts and the decomposition numbers for each key level hdnum_i, the HModulusSelection algorithm chooses the set of RNS moduli for each hierarchical special modulus. We design the ModulusSelection algorithm to minimize the bit-length of each special modulus, and it is used as a subroutine algorithm in the HModulusSelection algorithm as in Algorithm 3. The value maxmod is the maximum bit-length that we can use for each RNS primes, and this value is usually 60 when we use 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 Appendix B.

Theorem 3.1. *Algorithm 2 outputs the list of indices* $I = \{u_0 = 0, u_1, \cdots, u_{\mathsf{dnum}-1}\}$ *such that* $0 = u_0 < u_1 < \cdots < u_{\mathsf{dnum}-1} \le L$ *and minimizes the following formula*

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

Algorithm 2: ModulusSelection

Input: A list of modulus $\{q_0, \cdots, q_L\}$, decomposition number dnum

Output: Log of minimum special modulus $\log P$, the list of boundary indices $I = \{u_0 = 0, u_1, \cdots, u_{\mathsf{dnum}-1}\}$

```
if dnum = 1 then
    return Σ_{i=0}^{L} log q_i and {0}
else
    Find j minimizing the value
      | Σ_{i=0}^{j} log q_i − ((dnum−1)/dnum) Σ_{i=0}^{L} log q_i |.
    Perform ModulusSelection with {q_0, ··· , q_j} and
      dnum − 1 to output u_j and I_j.
    v_j ← Σ_{i=j+1}^{L} log q_i
    if u_j = v_j then return u_j and I_j ∪ {j + 1} ;
    else if u_j > v_j then
        while u_j ≥ v_j do
            j ← j − 1
            Perform ModulusSelection with {q_0, ··· , q_j}
              and dnum − 1 to output u_j and I_j.
            v_j ← Σ_{i=j+1}^{L} log q_i
        end
        if u_{j+1} > v_j then return v_j and I_j ∪ {j + 1} ;
        else return u_{j+1} and I_{j+1} ∪ {j + 2} ;
    else
        while u_j ≤ v_j do
            j ← j + 1
            Perform ModulusSelection for {q_0, ··· , q_j} and
              dnum − 1 to output u_j and I_j.
            v_j ← Σ_{i=j+1}^{L} log q_i
        end
        if u_j > v_{j−1} then return v_{j−1} and I_{j−1} ∪ {j} ;
        else return u_j and I_j ∪ {j + 1} ;
    end
end
```

We set the notation for the modulus as follows. The list of moduli for ciphertexts is denoted as $C = \{q_0, \cdots, q_L\}$, 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 $C \cup \bigcup_{j=0}^{\ell} \mathcal{B}_j$. The list of moduli for $C \cup \bigcup_{j=0}^{\ell} \mathcal{B}_j$ can be denoted as $\{q_0, \cdots, q_{L_\ell}\}$. Let $I_\ell = \{u_{\ell,0} = 0, u_{\ell,1}, \cdots, u_{\ell,\mathsf{hdnum}_\ell - 1}\}$ be the list of the indices of boundary position derived by HModulusSelection. Then, $Q_{\ell,i}$ denotes $\prod_{t=u_{\ell,i}}^{u_{\ell,i+1}-1} q_t$ for $0 \le \ell \le k-1$ and $0 \le i \le \mathsf{hdnum}_\ell - 1$ and $\hat{Q}_{\ell,i}$ denotes $\prod_{j \ne i} Q_{\ell,j}$.

Algorithm 3: HModulusSelection

Input: A list of modulus for ciphertexts $C = \{q_0, \cdots, q_L\}$, hierarchical decomposition number hdnum_ℓ for each ℓ, $0 \le \ell \le k-1$, the maximum bit-length of RNS modulus maxmod

Output: The lists of hierarchical special modulus $\mathcal{B}_\ell$ for each ℓ, $0 \le \ell \le k-1$ and the lists of boundary indices I_ℓ for each ℓ, $0 \le \ell \le k-1$

for $\ell = 0$ **to** $k-1$ **do**
  Perform ModulusSelection for $C \cup \bigcup_{j=0}^{\ell-1} \mathcal{B}_j$ and hdnum_ℓ to output $\log P_\ell$ and I_ℓ.
  $m_\ell \leftarrow \left\lceil \frac{\log P_\ell}{\mathsf{maxmod}} \right\rceil$
  Sample m_ℓ primes with a bit-length of $\log P_\ell / m_\ell$ and insert them to $\mathcal{B}_\ell$.
end

3.2 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 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 $\mathsf{gk}_r^{(k-1)} = \{\mathsf{gk}_{r,i}^{(k-1)}\}_{i=0,\cdots,\mathsf{hdnum}_{k-1}-1}$, where $\mathsf{gk}_{r,i}^{(k-1)} = (b_{r,i}^{(k-1)}, a_{r,i}^{(k-1)}) \in R^2_{Q_{k-1}P_{k-1}}$ such that $a_{r,i}^{(k-1)} \leftarrow R_{Q_{k-1}P_{k-1}}$ 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 $C \cup \bigcup_{j=0}^{k-1} \mathcal{B}_j$. 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.

3.3 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, we will first explain the operation GalToGal. 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 $m(X)$ [11]. We note that the Galois key for cyclic shift r is a set of ciphertexts $\mathsf{gk}_r^{(\ell)} = \{\mathsf{gk}_{r,i}^{(\ell)}\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1}$, where $\mathsf{gk}_{r,i}^{(\ell)} = (b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)}) \in R^2_{Q_\ell P_\ell}$ 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 we perform the rotation operation with cyclic shift r' on $\mathsf{gk}_{r,i}^{(\ell)}$, the output is a ciphertext of the following polynomial,

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

This rotation operation requires an (r', ℓ') Galois key $\mathsf{gk}_{r'}^{(\ell')}$, where ℓ' higher than ℓ. If we define this output as $\mathsf{gk}_{r+r',i}^{(\ell)}$, the set $\mathsf{gk}_{r+r'}^{(\ell)} = \{\mathsf{gk}_{r+r',i}^{(\ell)}\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1}$ is a valid $(r + r', \ell)$ Galois key. We will call this operation GalToGal, as shown in Algorithm 4. The following theorem shows the correctness of GalToGal operation, which will be proved in Appendix C.

Theorem 3.2. *The output of Algorithm 4 is a valid Galois key for the rotation operation for cyclic shift $r + r'$.*

Next, we will describe the operation PubToGal. 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, we can think of a formal shift-0 Galois key. If a shift-0 Galois key can be generated from a public key, a shift-r' Galois key can be generated by adding a GalToGal operation to the shift-0 Galois key for cyclic shift r'. By definition, the shift-0 Galois key should be the form of $\mathsf{gk}_0^{(\ell)} = \{\mathsf{gk}_{0,i}^{(\ell)}\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1}$, where $\mathsf{gk}_{0,i}^{(\ell)} = (b_{0,i}^{(\ell)}, a_{0,i}^{(\ell)}) \in R^2_{Q_\ell P_\ell}$ 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 $(b, a) \in R^2_{Q_{k-1}}$, we first reduce the public key to $(b', a') = (b \mod Q_\ell P_\ell, a \mod Q_\ell P_\ell) \in R^2_{Q_\ell P_\ell}$ by simply extracting values for corresponding RNS moduli. Then, we set as $b_{0,i}^{(\ell)} = b'$ and $a_{0,i}^{(\ell)} = a' + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}$. Then, we can have $b_{0,i}^{(\ell)} = -a_{0,i}^{(\ell)} \cdot s + e_{0,i}^{(\ell)} + P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}} \cdot s$. If we define $(b_{0,i}^{(\ell)}, a_{0,i}^{(\ell)})$ as $\mathsf{gk}_{0,i}^{(\ell)}$, the set $\mathsf{gk}_0^{(\ell)} = \{\mathsf{gk}_{0,i}^{(\ell)}\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1}$ is a valid formal $(0, \ell)$ Galois key. Then we can generate a shift-r Galois key by performing a GalToGal operation on it with the (r, ℓ) Galois key.

In addition, we can optimize the operations further by combining the decomposition processes in the key-switching operation. Trivially, the decomposition process is performed hdnum_ℓ 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 [3], 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, we perform the decomposition process 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 5. The correctness of this optimization is shown in the following theorem, which will be proved in Appendix D, 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 $\mu \leq \mathsf{hdnum}_{\ell'}$

Theorem 3.3. *The output of Algorithm 5 is a valid Galois key for the rotation operation for cyclic shift r.*

Algorithm 4: GalToGal

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

Output: An $(r+r', \ell)$ Galois key, $\mathsf{gk}_{r+r'}^{(\ell)} = \{b_{r+r',i}^{(\ell)}, a_{r+r',i}^{(\ell)}\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1} \in (R^2_{Q_\ell P_\ell})^{\mathsf{hdnum}_\ell}$

for $i = 0$ **to** $\mathsf{hdnum}_\ell - 1$ **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 $\mathsf{gk}_{r'}^{(\ell')}$.
end
return $\{(b_{r+r',i}^{(\ell)}, a_{r+r',i}^{(\ell)})\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1}$

Algorithm 5: PubToGal

Input: A public key $(b, a) \in R^2_{Q_{k-1}}$, a (r, ℓ') Galois key, $\mathsf{gk}_r^{(\ell')} = \{b_{r,j}^{(\ell')}, a_{r,j}^{(\ell')}\}_{j=0,\cdots,\mathsf{hdnum}_{\ell'}-1} \in (R^2_{Q_{\ell'} P_{\ell'}})^{\mathsf{hdnum}_{\ell'}}$, and the key level ℓ

Output: A (r, ℓ) Galois key, $\mathsf{gk}_r^{(\ell)} = \{b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)}\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1} \in (R^2_{Q_\ell P_\ell})^{\mathsf{hdnum}_\ell}$

$(b', a') \leftarrow ([b(X^{5^r})]_{Q_\ell P_\ell}, [a(X^{5^r})]_{Q_\ell P_\ell}) \in R^2_{Q_\ell P_\ell}$
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 $0 \leq j \leq \mu - 2$ 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** $\mathsf{hdnum}_\ell - 1$ **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
      $(\bar{b}, \bar{a}) \leftarrow (\bar{b}, \bar{a}) + (a_j + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}) \cdot ([b_{r,j}^{(\ell')}]_{P_{\ell'} Q_{\ell+1}}, [a_{r,j}^{(\ell')}]_{P_{\ell'} Q_{\ell+1}})$
    end
  end
  $(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)}) \leftarrow (\lfloor P_{\ell'}^{-1} \cdot \bar{b} \rceil, \lfloor P_{\ell'}^{-1} \cdot \bar{a} \rceil) \in R^2_{Q_{\ell+1}} = R^2_{P_\ell Q_\ell}$
  $b_{r,i}^{(\ell)} \leftarrow b_{r,i}^{(\ell)} + b'$
end
return $\{(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)})\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1}$

3.4 Galois Key Generation in the Lower Key Level

We can generate the desired level-ℓ Galois keys with only the public key and the Galois keys in the key level higher than ℓ through GalToGal and PubToGal described in Algorithm 6. We assume 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 we deal with the case when only one level-ℓ Galois key is generated. To generate the (r, ℓ) Galois key, we first perform the operation PubToGal with the shift-r_0 Galois key and the public key. Then we perform a GalToGal operation 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 6.

We usually have to generate a bundle of Galois keys rather than only one Galois key for a specific service. We will deal with the more efficient method for the case when we need to make a set of Galois keys at once in Section 4.

3.5 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 us nothing beyond the existing information [27]. 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, we only need 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 3.2, 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 $(R_{Q_{k-1}P_{k-1}})^2)^{\mathsf{hdnum}_{k-1}}$. 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, we can be given the maximum modulus bit length to guarantee the security level λ [3, 10], and the bit-length of $Q_{k-1}P_{k-1}$ should not exceed this bit length.

Algorithm 6: GalKeyGen for one Galois key

Input: A public key $(b, a) \in R^2_{Q_{k-1}}$, a set of Galois keys $\mathcal{G}_{\mathcal{U}_\ell} = \{\mathsf{gk}_r^{(\ell_r)} = \{(b_{r,i}^{(\ell_r)}, a_{r,i}^{(\ell_r)})\}_{i=0,\cdots,\mathsf{hdnum}_{\ell_r}-1} \in (R^2_{Q_{\ell_r}P_{\ell_r}})^{\mathsf{hdnum}_{\ell_r}} | r \in \mathcal{U}_\ell\}$ for cyclic shift generator set $\mathcal{U}_\ell$ in the key level higher than ℓ, and a cyclic shift $r = \sum_{u=0}^{t-1} r_u$ for $r_u \in \mathcal{U}_\ell$

Output: An (r, ℓ) Galois key, $\mathsf{gk}_r^{(\ell)} = \{b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)}\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1} \in (R^2_{Q_\ell P_\ell})^{\mathsf{hdnum}_\ell}$

$\{(b_{r_0,i}^{(\ell)}, a_{r_0,i}^{(\ell)})_{i=0,\cdots,\mathsf{hdnum}_\ell}\} \leftarrow$ PubToGal with the public key and the (r_0, ℓ_{r_0}) Galois key

for $h = 1$ **to** $t-1$ **do**

  $\{(b^{(\ell)}_{\sum_{j=0}^{h} r_j,i}, a^{(\ell)}_{\sum_{j=0}^{h} r_j,i})_{i=0,\cdots,\mathsf{hdnum}_\ell}\} \leftarrow$ GalToGal with $\{(b^{(\ell)}_{\sum_{j=0}^{h-1} r_j,i}, a^{(\ell)}_{\sum_{j=0}^{h-1} r_j,i})_{i=0,\cdots,\mathsf{hdnum}_\ell}\}$ and the (r_h, ℓ_{r_h}) Galois key

end

return $\mathsf{gk}_r^\ell = \{(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)})_{i=0,\cdots,\mathsf{hdnum}_\ell}\}$

4 EFFICIENT GENERATION METHOD OF GALOIS KEY SET

In the previous section, we 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, we 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. We 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 minimize the number of operations for generating these intermediate Galois keys by systematically organizing the generating sequence of the Galois keys.

4.1 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. We 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, we 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 our ordering of the Galois key generation problem as follows.

- We 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.
- We 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 our problem is to find a spanning arborescence with a minimum sum of edges, which is the goal of the minimum spanning arborescence problem. If we find a spanning arborescence in the graph, we 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 [12], 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, we 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, we 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, we can reduce this problem to the minimum

spanning tree problem, which can be solved by Prim's algorithm [29].

We note that this solution is not exactly the optimal solution since the insertion of additional nodes can reduce the operations further. If we set the nodes for all cyclic shifts (i.e., $\pm 1, \pm 2, \pm 3, \cdots$) in the graph, our 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, we 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.

4.2 Edge Weight for p-ary Galois Keys

We should consider a method to compute the edge of each graph, where we 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 $\mathcal{U}_\ell$, we suggest a specific example of key management system with $\mathcal{U}_\ell$ with power-of-p integers within the desired interval. We 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.

Algorithm 7: 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 $\sum_{i=0}^{\ell-1} t_i$

We consider the easier case, a set of positive power-of-p, in which the rotation graph is a directed graph. In this case, each edge can be computed as follows. First, we 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 7 without proof.

Next, we consider the case of power-of-p integers with both signs in which the rotation graph is an undirected graph as Section 4.1. 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, we 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 $pr_1 \le r < p(r_1+1)$ 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 8. To help understanding these reduction to the graph, we depict the toy example of Galois key graph in Appendix E.

Algorithm 8: 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$(p, r+1) + (p(r+1) - |t|)\}$
  end
end

4.3 Hoisted Galois Key Generation

The previous subsections focus on the reduction of the number of GalToGal and PubToGal operations. In this subsection, we 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 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 [3, 15]. We 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 we want to generate d Galois keys, we can naively perform exactly d PubToGal operations or d GalToGal operations. As we stated in Section 3.3, 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, we postpone the process of automorphism in line 2 of Algorithm 4 or line 1 of Algorithm 5 to the last of the operations to combine the decomposition processes into one process. To maintain the functionality of the operation, we 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, we 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, we reduce the number of decomposition processes from $d \cdot \mathsf{hdnum}_\ell$ to hdnum_ℓ. The hoisted version of GalToGal and PubToGal operations are described in Algorithms 9 and 10. The whole generation algorithm is described in Algorithm 11. We use breath-first search when we search each node in the output arborescence. This search method is desirable for the hoisted generation of Galois keys.

Algorithm 9: HoistedGalToGal

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

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

for $i = 0$ **to** $\mathsf{hdnum}_\ell - 1$ **do**
  Decompose $a_{r,j}^{(\ell)}$ 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 $\alpha = 0$ **to** $d-1$ **do**
    $(\bar{b}, \bar{a}) \leftarrow (0,0) \in R^2_{P_\ell Q_\ell}$
    for $j \leftarrow 0$ **to** $\mu - 1$ **do**
      $(\bar{b}, \bar{a}) \leftarrow (\bar{b}, \bar{a}) + a_j \cdot ([b_{r'_\alpha,j}^{(\ell')}(X^{5^{-r'_\alpha}})]_{P_{\ell'} Q_{\ell+1}}, [a_{r'_\alpha,j}^{(\ell')}(X^{5^{-r'_\alpha}})]_{P_{\ell'} Q_{\ell+1}})$
    end
    $(b_{r+r'_\alpha,i}^{(\ell)}, a_{r+r'_\alpha,i}^{(\ell)}) \leftarrow (\lfloor P_{\ell'}^{-1} \cdot \bar{b} \rceil, \lfloor P_{\ell'}^{-1} \cdot \bar{a} \rceil) \in R^2_{Q_{\ell+1}}$
    $b_{r+r'_\alpha,i}^{(\ell)} \leftarrow b_{r+r'_\alpha,i}^{(\ell)} + b_{r,i}^{(\ell)}$
    $(b_{r+r'_\alpha,i}^{(\ell)}, a_{r+r'_\alpha,i}^{(\ell)}) \leftarrow (b_{r+r'_\alpha,i}^{(\ell)}(X^{5^{r'_\alpha}}), a_{r+r'_\alpha,i}^{(\ell)}(X^{5^{r'_\alpha}}))$
  end
end
return $\{b_{r+r'_\alpha,i}^{(\ell)}, a_{r+r'_\alpha,i}^{(\ell)}\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1}$ for $\alpha = 0, \cdots, d-1$

Algorithm 10: 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, $\mathsf{gk}_{r_\alpha}^{(\ell')} = \{(b_{r_\alpha,i}^{(\ell')}, a_{r_\alpha,i}^{(\ell')})\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1} \in (R^2_{Q_{\ell'} P_{\ell'}})^{\mathsf{hdnum}_{\ell'}}$ for $\alpha = 0, \cdots, d-1$, and the key level ℓ

Output: d (r_α, ℓ) Galois keys, $\mathsf{gk}_{r_\alpha}^{(\ell)} = \{b_{r_\alpha,i}^{(\ell)}, a_{r_\alpha,i}^{(\ell)}\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1} \in (R^2_{Q_\ell P_\ell})^{\mathsf{hdnum}_\ell}$ 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** $\mathsf{hdnum}_\ell - 1$ **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**
        $(\bar{b}, \bar{a}) \leftarrow (\bar{b}, \bar{a}) + (a_{\mu-1} + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{\bar{Q}_{\ell',\mu-1}}) \cdot ([b_{r_\alpha,j}^{(\ell')}(X^{5^{-r_\alpha}})]_{P_{\ell'} Q_{\ell+1}}, [a_{r_\alpha,j}^{(\ell')}(X^{5^{-r_\alpha}})]_{P_{\ell'} Q_{\ell+1}})$
      else
        $(\bar{b}, \bar{a}) \leftarrow (\bar{b}, \bar{a}) + (a_j + [P_\ell \cdot \hat{Q}_{\ell,i} \cdot [\hat{Q}_{\ell,i}^{-1}]_{Q_{\ell,i}}]_{Q_{\ell',j}}) \cdot ([b_{r_\alpha,j}^{(\ell')}(X^{5^{-r_\alpha}})]_{P_{\ell'} Q_{\ell+1}}, [a_{r_\alpha,j}^{(\ell')}(X^{5^{-r_\alpha}})]_{P_{\ell'} Q_{\ell+1}})$
      end
    end
    $(b_{r_\alpha,i}^{(\ell)}, a_{r_\alpha,i}^{(\ell)}) \leftarrow (\lfloor P_{\ell'}^{-1} \cdot \bar{b} \rceil, \lfloor P_{\ell'}^{-1} \cdot \bar{a} \rceil) \in R^2_{Q_{\ell+1}}$
    $b_{r_\alpha,i}^{(\ell)} \leftarrow b_{r_\alpha,i}^{(\ell)} + [b]_{Q_{\ell+1}}$
    $(b_{r_\alpha,i}^{(\ell)}, a_{r_\alpha,i}^{(\ell)}) \leftarrow (b_{r_\alpha,i}^{(\ell)}(X^{5^{r_\alpha}}), a_{r_\alpha,i}^{(\ell)}(X^{5^{r_\alpha}}))$
  end
end
return $\{b_{r_\alpha,i}^{(\ell)}, a_{r_\alpha,i}^{(\ell)}\}_{i=0,\cdots,\mathsf{hdnum}_\ell-1}$ for $\alpha = 0, \cdots, d-1$

5 SIMULATION RESULTS WITH RESNET MODELS

In this section, we 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 computing model, the server usually has high-performance computing resources, and the client has only a general-purpose personal computer. To simulate this environment, we 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, we 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. Recently, Lee et al. [23] proposed several techniques for minimizing homomorphic operations for ResNet models with the CKKS scheme. They 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 [8, 23] is performed with different slots for different layers, where we can require different Galois keys for the bootstrapping operation with different number of sparse slots. Also, the sparse-secret encapsulation method [4] is assumed to be used for the bootstrapping with the dense secret key with more reduced running time and higher precision. We use the baby-step giant-step algorithm for fully connected layers [8]. When using all of these methods with CKKS algorithms using $N = 2^{16}$ for the polynomial modulus degree, we found that 265 Galois keys for different cyclic shifts are required in the ResNet-20 model processing the CIFAR-10 dataset, and the detailed cyclic shifts actually needed are shown in Appendix F.

The CKKS scheme is used for the simulation, and the parameters of the CKKS scheme we use for the simulation are shown in Table 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

Algorithm 11: GalKeyGen

Input: A cyclic shift set $\mathcal{T}_\ell$ for the key level ℓ, a cyclic shift generator set $\mathcal{U}_\ell$ for the key level higher than ℓ, a set of Galois keys $\mathcal{G}_{\mathcal{U}_\ell}$ for cyclic shifts in $\mathcal{U}_\ell$ in the key level higher than ℓ, 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
$\mathcal{G}_{\mathcal{T}_\ell} \leftarrow \emptyset$
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 $\mathcal{G}_W = \{\mathrm{gk}_w^{(\ell)} | w \in W\}$ from (b, a) using PubToGal or HoistedPubToGal
  else
    Generate the set of Galois keys $\mathcal{G}_W = \{\mathrm{gk}_w^{(\ell)} | w \in W\}$ from $\mathrm{gk}_v^{(\ell)}$ using GalToGal or HoistedGalToGal
  end
  $\mathcal{G}_{\mathcal{T}_\ell} \leftarrow \mathcal{G}_{\mathcal{T}_\ell} \cup \mathcal{G}_W$
  Enqueue elements in W to Q.
end
return $\mathcal{G}_{\mathcal{T}_\ell}$

Table 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

generation. The Galois key generation with the GPU processor is implemented based on [18]. 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 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 2 and 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 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)

Table 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)

Table 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 we 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

Table 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

average if we 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 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 we 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 we do not use the hoisted method. If we 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 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, 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.

We 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 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.

6 CONCLUSION

For the privacy-preserving AIaaS with FHE in the client-server model, we 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, we 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.

We constructed this hierarchical Galois key generation system for the BFV and CKKS schemes using a key-switching operation applied to the public key. Also, we 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, we 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.

REFERENCES

[1] 2022. Lattigo v3. Online: https://github.com/tuneinsight/lattigo. EPFL-LDS, Tune Insight SA.

[2] Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. 2016. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *International Conference on Selected Areas in Cryptography*. Springer, 423–442.

[3] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *EUROCRYPT 2021*.

[4] Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2022. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. *Cryptology ePrint Archive* (2022).

[5] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO 2012*. Springer, 868–886.

[6] Gizem S Çetin, Hao Chen, Kim Laine, Kristin Lauter, Peter Rindal, and Yuhou Xia. 2017. Private queries on encrypted genomic data. *BMC Medical Genomics* 10, 2 (2017), 1–14.

[7] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved bootstrapping for approximate homomorphic encryption. In *EUROCRYPT 2019*. Springer, 34–54.

[8] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT 2018*. Springer, 360–384.

[9] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. A full RNS variant of approximate homomorphic encryption. In *Proceedings*

Table 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 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

of International Conference on Selected Areas in Cryptography (SAC). 347–368.
[10] Jung Hee Cheon, Minki Hhan, Seungwan Hong, and Yongha Son. 2019. A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret LWE. *IEEE Access* 7 (2019), 89497–89506.
[11] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT 2017*. 409–437.
[12] Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards B* 71, 4 (1967), 233–240.
[13] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).
[14] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of International Conference on Machine Learning (ICML)*. PMLR, 201–210.
[15] Shai Halevi and Victor Shoup. 2018. Faster homomorphic linear transformations in HElib. In *CRYPTO 2018*. Springer, 93–120.
[16] Kyoohyung Han and Dohyeong Ki. 2020. Better bootstrapping for approximate homomorphic encryption. In *Cryptographers' Track at the RSA Conference*. Springer, 364–390.
[17] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1209–1222.
[18] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. 2021. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs. *Cryptol. ePrint Arch., Tech. Rep. 2021/508* (2021).
[19] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A low latency framework for secure neural network inference. In *Proceedings of the 27th USENIX Security Symposium*. 1651–1669.
[20] Miran Kim and Kristin Lauter. 2015. Private genome analysis through homomorphic encryption. In *BMC Medical informatics and decision making*, Vol. 15. BioMed Central, 1–12.
[21] Miran Kim, Yongsoo Song, Baiyu Li, and Daniele Micciancio. 2020. Semi-parallel logistic regression for GWAS on encrypted data. *BMC Medical Genomics* 13, 7 (2020), 1–13.
[22] Ovunc Kocabas and Tolga Soyata. 2020. Towards privacy-preserving medical cloud computing using homomorphic encryption. In *Virtual and Mobile Healthcare: Breakthroughs in Research and Practice*. IGI Global, 93–125.
[23] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2021. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed convolutions. *Cryptology ePrint Archive, Report 2021/1688* (2021).
[24] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. 2022. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access* 10 (2022), 30039–30054.
[25] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. 2021. High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In *EUROCRYPT 2021*.
[26] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, HyungChul Kang, and Jong-Seon No. 2020. High-precision and low-complexity approximate homomorphic encryption by error variance minimization. *Cryptology ePrint Archive* (2020). accepted to *EUROCRYPT 2022*.
[27] Yehuda Lindell. 2017. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography* (2017), 277–346.
[28] Souhail Meftah, Benjamin Hong Meng Tan, Khin Mi Mi Aung, Lu Yuxiao, Lin Jie, and Bharadwaj Veeravalli. 2022. Towards high performance homomorphic encryption for inference tasks on CPU: An MPI approach. *Future Generation Computer Systems* (2022).
[29] Robert Clay Prim. 1957. Shortest connection networks and some generalizations. *The Bell System Technical Journal* 36, 6 (1957), 1389–1401.

A PRELIMINARIES

A.1 Notations

Let $\mathbb{Z}$, $\mathbb{R}$, and $\mathbb{C}$ denote the set of integers, the set of real numbers, and the set of complex numbers, respectively. Let $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$, and let R and R_p denote the rings $\mathbb{R}[X]/(X^N+1)$ and $\mathbb{Z}_p[X]/(X^N+1)$, respectively. χ denotes an error distribution with a small variance, such as a Gaussian distribution. For integer x and positive integer p, $[x]_p$ denotes the non-negative remainder r such that $x = pq + r$, where q is an integer and $0 \le r \le p-1$. For integer x coprime with p, $[x^{-1}]_p$ denotes the inverse element of $[x]_p$ in $\mathbb{Z}_p$.

A.2 BFV and CKKS Schemes

Fully homomorphic encryption, abbreviated as homomorphic encryption, is an encryption scheme 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 operations while encrypted. The definition was mitigated to include word-wise encryption schemes capable of arbitrary arithmetic operations for encrypted integers or complex number data. The homomorphic encryptions covered in this paper are BFV and CKKS schemes. These homomorphic encryption schemes support arithmetic operations with the 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, integers are encrypted, and in the case of CKKS, complex numbers are encrypted. Although each scheme has several variants, we will address the schemes with the following encoding and encryption methods, where $(b, a) \in R_Q^2$ in a ring-LWE sample such that $b = -a \cdot s + e$ for the secret key $s \leftarrow \chi$ and a noise $e \leftarrow \chi$.

- 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 $\mathbb{Z}_t$. We then obtain $m(X) \in R_t$ 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 $\mathbb{C}$. We then obtain $m(X) \in \mathbb{R}[X]/(X^N + 1)$ 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 $u, e_0, e_1 \leftarrow \chi$ and Δ is scaling factor that determines the precision of m.

We assume that the residue number system (RNS) variants of BFV and CKKS schemes [2, 9] are used. In this variant, 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 ModDown operations. The ModUp operation raises the modulus with remaining the remainder, and the 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 ModDwon operations are described in Algorithms 12 and 13, where we assume that each ring element is in the NTT form.

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, we only represent these pairs (b, a) for the encryption process as the public key in this paper. 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 paper, we only deal with the rotation operation for $m(X)$ corresponding to cyclic shift of the message vector. The detailed explanations of other operations can be found in [5, 9, 11, 13].

Algorithm 12: 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, ..., a_{σ-1}).
for i ← 0 to σ − 1 do
    ā_i ← a_i
    b_i ← a_i · [q̂_i^{-1}]_{q_i} ∈ R_{q_i}
end
for j ← 0 to τ − 1 do
    ã_j ← 0
    for i ← 0 to σ − 1 do
        ã_j ← ã_j + b_i · [q̂_i]_{p_j} ∈ R_{p_j}
    end
end
NTT operation to (ā_0, ..., ā_{σ-1}, ã_0, ..., ã_{τ-1}).
return (ā_0, ..., ā_{σ-1}, ã_0, ..., ã_{τ-1}) ∈ ∏_{i=0}^{σ-1} R_{q_i} × ∏_{j=0}^{τ-1} R_{p_i}
```

Algorithm 13: 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 ← 0 to σ − 1 do
    b̄_i ← ā_i + [⌊P/2⌋]_{q_i} ∈ R_{q_i}
end
for j ← 0 to τ − 1 do
    b̃_j ← ã_j + [⌊P/2⌋]_{p_j} ∈ R_{p_j}
end
(b̄'_0, ..., b̄'_{σ-1}, b̃'_0, ..., b̃'_{τ-1}) ← ModUp for (b̃_0, ..., b̃_{τ-1})
  from ∏_{j=0}^{τ-1} R_{p_i} to ∏_{i=0}^{σ-1} R_{q_i} × ∏_{j=0}^{τ-1} R_{p_i}
for i ← 0 to σ − 1 do
    a'_i ← [P^{-1}]_{q_i} · (b̄_i − b̄'_i) ∈ R_{q_i}
end
return (a'_0, ..., a'_{σ-1}) ∈ ∏_{i=0}^{σ-1} R_{q_i}
```

A.3 Key-Switching Operation and Galois Key

We now explain the key-switching operation [16] 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 we want to perform the key-switching operation switching the secret key from s to s'. The RNS moduli that we use for key-switching are Q_i for $i = 0, \cdots, \mathsf{dnum} - 1$ 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. We construct the key-switching key 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 14. Each RNS element is multiplied by the ciphertext having the corresponding RNS basis in the key-switching key and added with each other. Then, we divide the ciphertext and the modulus by the special modulus with the ModDown operation. The whole algorithm for key-switching operation is described in Algorithm 15. 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 14: 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.

1 **for** $i \leftarrow 0$ **to** $\delta - 1$ **do**
2 $\quad a_i \leftarrow$ ModUp for $[a]_{Q_i} \in R_{Q_i}$ from R_{Q_i} to R_{PQ}.
3 **end**
4 **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 we perform a deep homomorphic circuit with the bootstrapping operations. The value of dnum is selected in consideration of these trade-offs.

If we want to perform the rotation operation for cyclic shift r, the key-switching key for this operation can be constructed as above for $s' = s(X^{5^r})$. We will call this key-switching key 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 15.

Note that in this algorithm, we deal with a general case when the modulus $\bar{Q}$ of a ciphertext is a divisor of the maximum evaluation modulus Q. We can simply replace Q 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, we add the proof for this fact in the following theorem.

Theorem A.1. *Assume that*

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

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 $\mu \leq$ 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

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

is valid, we have

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

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

$$[b^{(i)} + a^{(i)} \cdot s]_{Q_j} = \begin{cases} [P]_{Q_i} \cdot s(X^{5^r}) + e_i & \text{if } i = j \\ e_i & \text{if } i \neq j \end{cases} \quad (1)$$

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

On the other hand, we consider the following ring element

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

where $\bar{Q}_i = Q_i$ for $0 \leq i \leq \mu - 2$ and $\hat{\bar{Q}}_i = \prod_{j=0, j\neq i}^{\mu-1} \bar{Q}_i$. Note that we have

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

If we perform the modular reduction 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, we have $[P \cdot \hat{\bar{Q}}_i \cdot [\hat{\bar{Q}}_i^{-1}]_{\bar{Q}_i} \cdot s(X^{5^r}) + e_i]_P = e_i$.

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

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

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. □

Algorithm 15: Key-Switching Operation [16]

Input: A key-switching key from s to s', $\mathsf{swk} = \{(b^{(i)}, a^{(i)})\}_{i=0,\cdots,\mathsf{dnum}-1} \in (R_{PQ}^2)^{\mathsf{dnum}}$ for $Q = \prod_{i=0}^{\mathsf{dnum}-1} Q_i$, 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 $\mu \le \mathsf{dnum}$.

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 $0 \le i \le \mu - 2$ 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**
  $(\bar{b}, \bar{a}) \leftarrow (\bar{b}, \bar{a}) + a_i \cdot ([b^{(i)}]_{P\bar{Q}}, [a^{(i)}]_{P\bar{Q}})$
end
$(b', a') \leftarrow (\lfloor P^{-1} \cdot \bar{b} \rceil, \lfloor P^{-1} \cdot \bar{a} \rceil) \in R_{\bar{Q}}^2$
$b' \leftarrow b' + b$
return (b', a')

A.4 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 [12], shown in Algorithm 16.

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 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 we will use Prim's algorithm [29] appropriate for the dense graph in this paper because we deal with a complete graph, shown in Algorithm 17.

Algorithm 16: Edmonds' Algorithm[12]

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
  $E' \leftarrow E' \cup \{(\pi(v), v)\}$
end
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 $w(v_1, v_c) = w(v_1, v_2) - w(\pi(v_2), v_2)$
    $\bar{E} \leftarrow (\bar{E} \backslash \{(v_1, v_2)\}) \cup \{(v_1, v_c)\}$
  end
  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 $w(v_c, v_2) = w(v_1, v_2)$
      $\bar{E} \leftarrow \bar{E} \backslash \{(v_1, v_2)\} \cup \{(v_c, v_2)\}$
    end
  end
  $\bar{G}' = (\bar{V}, \bar{E}') \leftarrow$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 $G' \leftarrow (V, E')$
end

Algorithm 17: Prim's Algorithm[29]

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 $\bar{E}$
  $V' \leftarrow V' \cup \{\bar{v}_2\}$
  $E' \leftarrow E' \cup \{\bar{e}\}$
end
return $G' = (V, E')$

B PROOF OF THEOREM 3.1

To prove Theorem 3.1, we need the following three lemmas. After proving the lemmas, we prove Theorem 3.1 using these lemmas.

Lemma B.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 we define $\min_{i\in I} f(i) = \infty$ for $I = \emptyset$ 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 we 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 we 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 $\alpha_m(j)$, we also have $\beta_j(u_{j+1,0} = 0, u_{j+1,1}, \cdots, u_{j+1,m-1}) \geq f(j)$. Therefore, we have $\alpha_m(j) \leq \alpha_m(j+1)$, which contradicts the assumption. □

Lemma B.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 B.3. *For positive numbers $v_0, \cdots, v_L$ and integer $m > 1$, the function $\alpha_m(j)$ is defined as Lemma B.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

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

If a set A is decomposed into a union of disjoint sets A_i for index set I, that is, $A = \bigsqcup_{i\in I} A_i$, we 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 $\alpha_m(L) = \min_{0\leq u\leq L} \alpha_{m,u}(L)$. We want to show that for all u, $0 \leq u \leq L$, we have

$$\alpha_{m,u}(L) = \max \left\{ \alpha_{m-1}(u-1), \sum_{i=u}^{L} v_i \right\}. \tag{3}$$

If we fix u in (3), the term $\sum_{i=u}^{L} v_i$ in the right side of (3) is a constant. We 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 (3), we know $\alpha_{m,u}(L) \geq \sum_{i=u}^{L} v_i$ since the term in the min function in (2) 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 (2) 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 (3) is $\sum_{i=u}^{L}$, (3) 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 (2) for $j = L$ holds even if we 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 (3) is $\alpha_{m-1}(u-1)$, (3) holds.

□

Theorem 3.1. *Algorithm 2 outputs the list of indices $I = \{u_0 = 0, u_1, \cdots, u_{\mathsf{dnum}-1}\}$ such that $0 = u_0 < u_1 < \cdots < u_{\mathsf{dnum}-1} \leq L$ and minimizes the following formula*

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

Proof. We now prove Theorem 3.1 by induction with dnum.

(1) dnum = 1: The equality trivially holds.

(2) $\mathsf{dnum} > 1$: Assume that the equality holds for $\mathsf{dnum} - 1$. If we set $v_i = \log q_i$, $\alpha_{\mathsf{dnum}-1}(j)$ is a monotonously increasing function. If we 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 $\alpha_{\mathsf{dnum}-1}$ and γ satisfy the conditions in Lemma B.2, there is an integer $\ell \in \mathbb{Z} \cap [0, L)$ such that $\alpha_{\mathsf{dnum}-1}(n) \leq \gamma(n)$ for $n \in [0, \ell]$, $\alpha_{\mathsf{dnum}-1}(n) > \gamma(n)$ for $n \in [\ell+1, L]$, and the minimum value of $h(n) = \max\{\alpha_{\mathsf{dnum}-1}(n), \gamma(n)\}$ is $\min\{\gamma(\ell), \alpha_{\mathsf{dnum}-1}(\ell+1)\}$.
If $j = j_0$ in line 4 of Algorithm 2 satisfies $\alpha_{\mathsf{dnum}-1}(j_0) = \gamma(j_0)$, we have $j_0 = \ell$, and the value $\alpha_{\mathsf{dnum}-1}(j_0) = \gamma(j_0)$ is the minimum value of $h(n)$. If $\alpha_{\mathsf{dnum}-1}(j_0) > \gamma(j_0)$, we have $j_0 \geq \ell + 1$. We can find ℓ at the point when $\alpha_{\mathsf{dnum}-1}(j) \leq \gamma(j)$ first holds as j decreases from j_0 by 1. Then, the value $\min\{\alpha_{\mathsf{dnum}-1}(\ell+1), \gamma(\ell)\}$ is the minimum value of $h(n)$. If $\alpha_{\mathsf{dnum}-1}(j_0) < \gamma(j_0)$, we have $j_0 \leq \ell$. We can find $\ell' = \ell+1$ at the point when $\alpha_{\mathsf{dnum}-1}(j) > \gamma(j)$ first holds as j increases from j_0 by 1. Then, the value $\min\{\alpha_{\mathsf{dnum}-1}(\ell'), \gamma(\ell'-1)\} = \min\{\alpha_{\mathsf{dnum}-1}(\ell+1), \gamma(\ell)\}$ is the minimum value of $h(n)$.
For all cases, we can find the minimum value of $h(n)$. This minimum value is actually the minimum value of $f(u_0 = 0, \cdots, u_{\mathsf{dnum}-1})$ by Lemma B.3, which proves the theorem.

□

C PROOF OF THEOREM 3.2

Theorem 3.2. *The output of Algorithm 4 is a valid Galois key for the rotation operation for cyclic shift $r + r'$.*

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

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

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 we 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 4, we 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 we 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 4, 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,

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

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

To use the Galois key $\mathsf{gk}_{r'}^{(\ell')} \in (R^2_{Q_{\ell'}P_{\ell'}})^{\mathsf{hdnum}_{\ell'}}$ 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 $\mathsf{gk}_{r'}^{(\ell')}$.

□

D PROOF OF THEOREM 3.3

Theorem 3.3. *The output of Algorithm 5 is a valid Galois key for the rotation operation for cyclic shift r.*

Proof. We give the proof for $\ell' = \ell + 1$. The proof for $\ell' > \ell + 1$ is the same as the case of $\ell' = \ell + 1$ by Theorem A.1. A public key $(b, a) \in R^2_{Q_{k-1}}$ 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 we perform

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

as in line 1 of Algorithm 5, we 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 we decompose a' into a vector $(a_0, \cdots, a_{\mathsf{hdnum}_{\ell'}-1}) \in R^{\mathsf{hdnum}_{\ell'}}_{P_{\ell'}Q_{\ell'}}$ using ModUp opeation, we 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 [2], 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

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

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

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

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

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

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

If we 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 we denotes $(b_{r,i}^{(\ell)}, a_{r,i}^{(\ell)})$, we 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 we 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,

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

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

□

E TOY EXAMPLE OF GALOIS KEY GRAPH

To help understanding the graph-theoretic algorithms for Galois key generation in Section 4, we depict the corresponding graph and the minimum spanning tree for $\mathcal{T}_\ell = \{1, 13, 16, 17, 19\}$ and $\mathcal{U}_\ell = \{\pm 1, \pm 2, \pm 4, \pm 8, \pm 16\}$ in Figure 2.

F REQUIRED GALOIS KEYS FOR RESNET MODELS

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.

- $\mathcal{T}_0^{\mathsf{ResNet-18}} = \{1, -1, 2, -2, 3, -3, 4, -4, 5, -5, 6, -6, 7, -7, 8, -8, -9, -10, -11, -12, -13, -14, -15, 16, -16, -17, -18, -19, -20, -21, -22, -23, 24, -24, -25, -26, -27, -28, -29, -30, -31, 32, -32, 39, 64, 78, 96, 117, 128, 156, 160, 192, 195, 221, 222, 224, -224, -225, -226, -227, -228, -229, -230, -231, -232, -233, 234, -234, -235,$

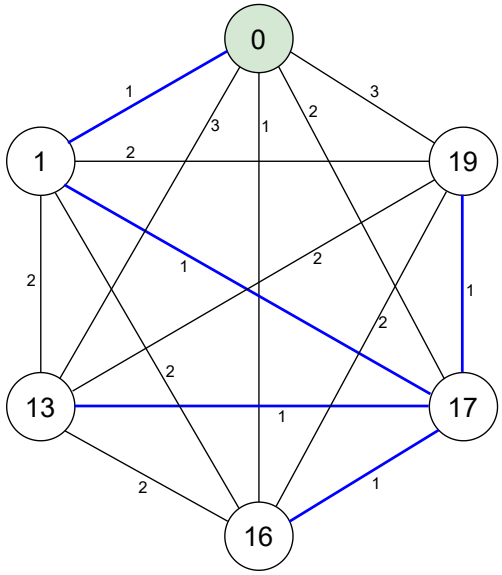


Figure 2: Galois key graph for $\mathcal{T}_\ell = \{1, 13, 16, 17, 19\}$ and $\mathcal{U}_\ell = \{\pm 1, \pm 2, \pm 4, \pm 8, \pm 16\}$.

-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, -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}