Ph.D. DISSERTATION

Client-Aided Deep Neural Network on Fully Homomorphic Encryption without Bootstrapping and Attack Algorithm for a Keystore-based Key Generation

클라이언트를 이용하여 부트스트래핑을 제거한
완전동형암호상의 딥 뉴럴 네트워크와 키 스토어 기반
키 생성 방식 공격 알고리즘

BY

CHAE SEUNGJAE

AUGUST 2023

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

Ph.D. DISSERTATION

Client-Aided Deep Neural Network on Fully Homomorphic Encryption without Bootstrapping and Attack Algorithm for a Keystore-based Key Generation

클라이언트를 이용하여 부트스트래핑을 제거한
완전동형암호상의 딥 뉴럴 네트워크와 키 스토어 기반
키 생성 방식 공격 알고리즘

BY

CHAE SEUNGJAE

AUGUST 2023

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

Client-Aided Deep Neural Network on Fully Homomorphic Encryption without Bootstrapping and Attack Algorithm for a Keystore-based Key Generation

클라이언트를 이용하여 부트스트래핑을 제거한 완전동형암호상의 딥 뉴럴 네트워크와 키 스토어 기반 키 생성 방식 공격 알고리즘

지도교수 노 종 선

이 논문을 공학박사 학위논문으로 제출함

2023년 8월

서울대학교 대학원

전기 정보 공학부

채 승 재

채승재의 공학박사 학위 논문을 인준함

2023년 8월

Abstract

In this dissertation, two main contributions are given as: i) Client-aided deep neural network on fully homomorphic encryption(FHE) without bootstrapping using communication cost in the client-server model. ii) attack algorithm for a keystore-based secure key generation and management

First, client-aided privacy-preserving machine learning on fully homomorphic encryption without bootstrapping is proposed. Bootstrapping which is the heaviest computation in homomorphic encryption consumes almost 70% of the total computation in homomorphic encryption. In order to avoid this problem, multi-party computation(MPC) based privacy-preserving machine learning(PPML) was introduced. However, this method cannot use pre-trained parameters due to the hardness to use exact Rectified Linear Unit (ReLU) in PPML. Recently, using minimax approximate polynomials for sign functions for HE-PPML[2, 3, 4], MPC-PPML without bootstrapping can be implemented with communication cost in the client-server model. Since HE-friendly networks do not use non-arithmetic functions like ReLU or max pooling, their inferences are light and fast. However, low classification accuracy and training the data are very difficult, and thus using pre-trained parameters is a very significant issue in the PPML. Thus I propose a method that improves DELPHI [7] algorithm to inference with pre-trained parameters on homomorphic encryption. In terms of the computation time and communication cost the proposed method has better performance compared to DELPHI, the previous work of MPC-based inference schemes.

Second, a new attack algorithm is proposed for a secure key generation and management method introduced by Yang and Wu. It was previously claimed that the key generation method of Yang and Wu[46] using a keystore seed was information-theoretically secure and could solve the long-term key storage problem in cloud systems, thanks to the huge number of secure keys that the keystore seed can generate.

Their key generation method, however, is considered to be broken if an attacker can recover the keystore seed. In this dissertation, I propose an attack algorithm to reconstruct the keystore seed of the Yang–Wu key generation method from a small number of collected keys. For example, when $t = 5$ and $l = 2^7$, it was previously claimed that more than 2^{53} secure keys could be generated, but the proposed attack algorithm can reconstruct the keystore seed based on only 84 collected keys. Hence it turns out that the Yang–Wu key generation method is not information-theoretically secure when the attacker can gather multiple keys and a critical amount of information about the keystore seed is leaked.

keywords: Fully homomorphic encryption (FHE), multi-party computation (MPC), privacy-preserving machine learning(PPML), Cheon-Kim-Kim-Song(CKKS) scheme, cryptography, ciphertext refresh, deep neural network, bootstrapping, information-theoretically secure, key generation, key management, keystore seed, data privacy
student number: 2017-22828

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Background

When performing a certain operation, lots of researches have been conducted only theoretically on technology for performing an operation while keeping a user's input secretly. This is because previous encryption technologies placed more importance on corporate security than personal information protection. However, due to the recent development of studies, users have begun to pay attention to their personal data privacy and research related to anonymization such as homomorphic encryption and multiparty computations are receiving a lot of attention. Machine learning has been established as the core of recent research, and is being used widely in lots of applications. As personal data privacy becomes more important, the need to preserving privacy in machine learning is emphasized. Privacy-preserving machine learning(PPML), is one of the hottest topics in security and cryptography and is being studied rapidly in recent years.

Data storage and transmission have been frequently used in recent public cloud systems. It is important to use secure keys in the cloud system, because users using a password can be vulnerable to dictionary attacks [43]. It is well known that secure keys reveal less user information than the password method. Thus, secure keys have been used in various fields such as file encryption, access to virtual private networks, and user authentication [44]. However, conventional key generation methods have many problems in terms of long-term file management, where each file should be independently encrypted with random secure keys since it has the characteristics of long-term file storage and frequent user access. Otherwise, cloud systems are not secure for ciphertext-only attack or chosen-plaintext attack [45]. To make one-key-for-one-file secure encryption for long-term data protection, a new secure key generation method using the keystore seed was proposed in [46] claiming that their method could make many information-theoretically ϵ-secure keys. In this dissertation, I propose a new method to break their key generation in [46] by reconstructing the keystore seed using a small number of collected keys.

1.2 Overview of Dissertation

This dissertation is organized as follows. In Chapter 2, the preliminaries of fully homomorphic encryption(FHE), multi-party computation(MPC), privacy-preserving machine learning, and secure key generation method using keystore seed are introduced. Section 2.1 presents basic descriptions of privacy-preserving machine learning. In Section 2.2, key generation and management based on keystore seed are illustrated and their information-theoretically secureness is discussed.

In Chapter 3, a new construction method for bootstrapping replacement using communication cost in the client-server model is proposed. Section 3.1 introduces the basic concepts of privacy-preserving machine learning. In Section 3.2, previous works using fully homomorphic encryption and multi-party computation of deep convolutional neural networks are given. A new construction method of client-aided deep neural network on fully homomorphic encryption without bootstrapping in the client-server model using communication cost is introduced in Section 3.3. In Section 3.4, all simulation results reducing communication cost and inference time without bootstrapping are given and compared to the previous schemes.

In Chapter 4, an attack algorithm for a keystore-based secret key generation method is proposed. Section 4.1 introduces the basic concepts of key generation and management using keystore seed. Problems of generating and management using exclusive or keys are introduced in Section 4.2. In Section 4.3, a brief formulation of a linear attack on key generation and management using keystore seed is given. However, modified with hashed keys still have information-theoretic weaknesses and that is given in Section 4.4.

Chapter 2

Preliminaries

2.1 Privacy-Preserving Machine Learning

2.1.1 Fully Homomorphic Encryption

Fully homomorphic encryption is an advanced encryption technology that empowers arithmetic processing, such as search and statistical operations, to be performed on encrypted data without requiring decryption. This remarkable capability has sparked significant research interest, particularly as a solution to prevent information leakage in the context of cloud computing, which has gained widespread usage in recent times. The fundamental idea behind FHE is to enable users to encrypt their data and transmit it to a server, where computations can be carried out on the encrypted data. The server then sends the processing results back to the user, who subsequently decrypts the returned ciphertext to retrieve the final outcome. Notably, FHE allows the execution of various operations, such as addition and multiplication, on the ciphertext without compromising the confidentiality of the underlying information. This property of supporting a broad range of operations makes it ”fully” homomorphic encryption. A brief description of fully homomorphic encryption can be found in Figure 2.1.

However, a notable challenge in FHE is the accumulation of errors during the course of operations. As computations progress, the size of the error tends to increase, which can impact the accuracy and reliability of the final results. In scenarios where only one of the two operations (addition or multiplication) is feasible, or the number of operations that can be performed is limited, the encryption scheme is referred to as ”Somewhat Homomorphic Encryption” (SHE). SHE provides a more restricted form of homomorphic computation compared to FHE, but it still offers valuable privacy-preserving capabilities in scenarios where fully homomorphic encryption may not be necessary or feasible.

In 2009, Gentry developed the concept of bootstrapping and showed the possibility of fully homomorphic encryption that can continue to perform operations.[11] Representative fully homomorphic encryption schemes are the fast Fully Homomorphic Encryption over the Torus(TFHE) algorithm [15] and the Cheon-Kim-Kim-Song(CKKS) algorithm [13].

When comparing the advantages and disadvantages of the two algorithms, CKKS and TFHE, it becomes evident that each algorithm has its unique strengths and weaknesses. CKKS is particularly notable for its ability to support operations on both real and complex data, making it highly versatile in practical applications. Additionally, CKKS offers a high level of utility, enabling various computations to be performed on encrypted data. On the other hand, TFHE exhibits its advantages in terms of accelerating bootstrapping operations, making it well-suited for non-linear functions and bit operations. Table 2.1 shows the comparison between fully homomorphic encryption schemes.

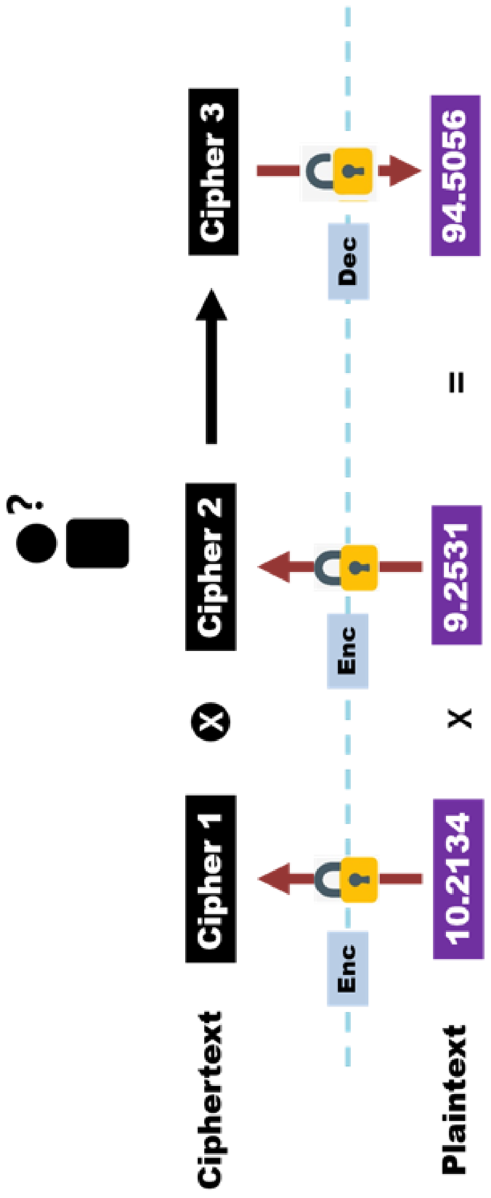


Figure 2.1: Fully homomorphic encryption.

Table 2.1: Fully homomorphic encryption schemes

	CKKS	BGV,BFV	TFHE
Data	Real and complex	Finite integer ring	Binary
Operation	Arithmetic circuit	Arithmetic circuit	Boolean circuit
Decryption	Not exact	Exact	Exact
Packing	Yes	Yes	No
Rescaling	Yes	No	Yes

However, alongside their advantages, both algorithms also present certain limitations. The CKKS algorithm introduces an error that ensures the security of the data, treating it as an approximation error during computations. While this error is essential for maintaining the confidentiality of the encrypted data, it can result in a drawback when executing successive operations because the bootstrapping process is required, which is time-consuming.

In the case of TFHE, one disadvantage lies in its limited support for SIMD properties compared to other homomorphic encryption algorithms. Furthermore, TFHE requires bootstrapping for each gate, resulting in a constraint that only one-bit operation can be performed at a time. This characteristic poses a challenge in scenarios where multiple concurrent operations are necessary.

It is worth noting that CKKS is currently garnering significant attention among fully homomorphic encryption schemes because it is based on a lattice-based hard problem, CKKS is designed to withstand potential attacks from quantum computing adversaries. In terms of security, the noise introduced in CKKS can be understood as an approximate encryption scheme, with the decryption process aiming to recover the original message while accounting for this noise.

Then, I explain the CKKS algorithm in detail. First, the CKKS encryption process is given as :

- Let $R_q = Z_q[X]/(X^N + 1)$
- A ring-LWE sample $(b, a) \in R_Q^2$ is a public key such that a is uniformly sampled and $b + a \cdot s = e$ for small e
- Ciphertext $ct = (b', a') = v \cdot (b, a) + (m + e_0, e_1) \in R_Q^2$ v, e_1, e_2 are a Gaussian random polynomials

Next, rescaling performed after multiplications to scale a real number data is explained as follows :

- Multiplying two real numbers m_1, m_2, the scaling factor is required to be devided by Δ .
- $(\Delta \cdot m_1) \cdot (\Delta \cdot m_1) = \Delta^2 \cdot m_1 \cdot m_2 => \Delta \cdot m_1 \cdot m_2$

All of the above processes are summarized in the Figure 2.2.

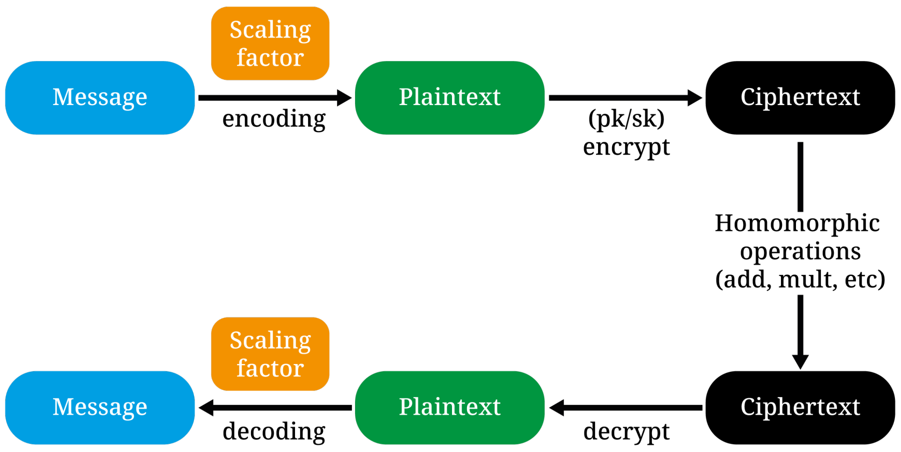


Figure 2.2: CKKS algorithm.

There are four representative homomorphic operations as below.

- Addition : $Enc(x_0, x_1, \ldots, x_n) \oplus Enc(y_o, y_1, \ldots, y_n) = Enc(x_0 + y_o, x_1 + y_1, \ldots, x_n + y_n)$
- Scalar multiplication : $(x_0, x_1, \ldots, x_n) \cdot Enc(y_o, y_1, \ldots, y_n) = Enc(x_0 \cdot y_o, x_1 \cdot y_1, \ldots, x_n \cdot y_n)$
- Multiplication : $Enc(x_0, x_1, \ldots, x_n) \cdot Enc(y_o, y_1, \ldots, y_n) = Enc(x_0 \cdot y_o, x_1 \cdot y_1, \ldots, x_n \cdot y_n)$
- Rotation(by k) : $Enc(x_0, x_1, \ldots, x_n) -> Enc(x_k, \ldots, x_n, x_0 \ldots, x_{k-1})$

For CKKS scheme, several definitions are needed as follows :

- Ciphertext level: possible number of multiplications
- Depth: maximum number of consecutive multiplications
- Ciphertext modulus: the capacity of multiplicative depth

As operations on the ciphertext has proceeded, the ciphertext modulus becomes small and the modulus should be raised again through the bootstrapping process [23]. For large-depth applications need this process is needed to continue the operations. Figure 2.3 shows the CKKS bootstrapping procedure.

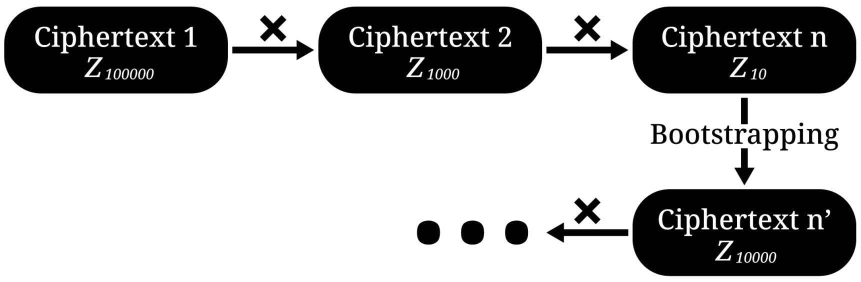


Figure 2.3: CKKS bootstrapping.

Residue number system(RNS)-CKKS is a method introduced [9] and is a method using the residue number system for the CKKS scheme. The original CKKS scheme needs integers with hundreds or thousands of bits and large integers can be represented by a tuple of 64-bit integers with the chinese remainder theorem. In these cases, it does not need any multi-precision integer library and is faster than the original CKKS scheme. Each multiplication removes one 64-bit RNS integer with a rescaling procedure.

2.1.2 Multi-Party Computation

Multi-party computation is a powerful technology that allows multiple participants or users to collaboratively compute a common function by inputting their own secret information. The concept of MPC was originated from the need for certain groups to perform computation tasks based on private inputs, where a single entity access to all participants' inputs in order to perform the calculation. This becomes a critical concern in scenarios where trust among entities is limited. To address this problem, Yao's two-party computation [1] was introduced as the initial solution to the problem.

MPC offers significant advantages, primarily in terms of privacy preservation without the need for external intervention, which has led to extensive ongoing research in this field. One of the key distinctions between fully homomorphic encryption and multi-party computation lies in the communication aspect among users. In fully homomorphic encryption, most calculations are performed by a central operator, except for initial and final communications. However, when it comes to a server performing fully homomorphic encryption, it becomes burdened with high-complexity calculations that involve a substantial amount of data. Figure 2.4 shows the multi-party computation between users at a glance.

On the other hand, multi-party computation involves relatively straightforward operations, but a notable challenge arises due to the requirement for communication of a large amount of data. To address the problems on FHE and MPC, researchers have ex-

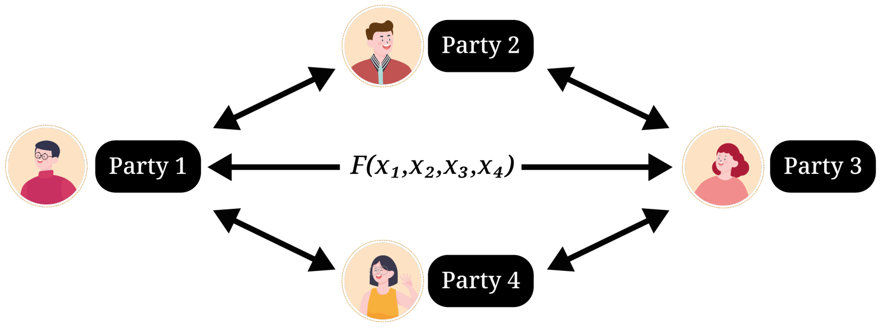


Figure 2.4: Multi-party computation.

plored combining fully homomorphic encryption with multi-party computations. This integration allows for the concealment of one's own data and facilitates the execution of desired calculations. It also enables the direct implementation of desired operations as circuit stages. Someone considers this combination as a transitional technology that is, MPC itself. Nonetheless, active research is being conducted in this area due to its performance advantages of PPML on the FHE in the client-server system.

Overall, MPC offers a promising approach to collaborative computation, ensuring privacy and enabling secure operations among multiple participants. While challenges and constraints exist, ongoing research aims to overcome these limitations and further enhance the efficiency and effectiveness of MPC.

Now I explain SPDZ [14], one of the famous multi-party computation algorithms as follows :

- Assume that n parties of which $n-1$ could be malicious
- There exists a global secret key $\alpha \in F_p$
- Each party i holds α_i such that $\alpha = \alpha_1 + \alpha_2 + \cdots + \alpha_n$

Under these assumptions, a secret value $x \in F_p$ is shared between the parties as follows :

- Each party i holds a data share x_i
- Each party i holds a MAC share $\gamma_i(x)$
- $x = x_1 + \cdots + x_n$ and $\alpha \cdot x = \gamma_1(x) + \cdots + \gamma_n(x)$

Such a sharing is denoted by $[x]$ and assumes that the player's inputs are shared using above shared rules. If someone wants to add two shared secret values $[x]$ and $[y]$, compute the result $[z]$ as follows :

- $z_i = x_i + y_i$
- $\gamma_i(z) = \gamma_i(x) + \gamma_i(y)$
- $z = \sum z_i = x + y$
- $\alpha \cdot z = \sum \gamma_i(z) = \alpha \cdot (x + y)$

Partially open means that shared $[x]$ reveals x_i but not reveals MAC share. These definitions are used for secret value multiplications. To multiply two shared values $[x]$ and $[y]$, the following procedure is needed as follows :

- Preprocessing to make a triple (a, b, c) which satisfy $c = a \cdot b$
- Partially opens $[x] - [a]$ and $[y] - [b]$
- Locally computes the linear function $[z] = [c] + (x - a) \cdot [b] + (y - b) \cdot [a] + (x - a) \cdot (y - b)$
- Rearranging the above expression, get $[z] = [x] \cdot [y]$

It is possible to verify that the operation performed through MAC authentication is correct. If both addition and multiplication are possible, all linear operations can be expressed in two operations, and thus SPDZ proceeds based on this.

2.1.3 Recent Research of Privacy-Preserving Machine Learning

Machine learning has experienced rapid advancements in recent years, surpassing human intelligence in certain domains. As a result, ML models are being swiftly adopted across various industries, thanks to their remarkable performance improvements. However, this progress has brought forth a critical concern – the issue of privacy. As ML services become more prevalent, ensuring the privacy of users' data has become a paramount consideration for artificial intelligence.

The rise of legal disputes surrounding privacy in ML services among governments, enterprises, and clients underscores the increasing social importance of privacy preservation. To foster the growth of the AI industry, it is crucial to implement ML systems efficiently while safeguarding user data against potential leaks. This way, clients can use ML services with confidence, bolstering the overall AI ecosystem.

Among the practical cryptographic tools for designing privacy-preserving AI systems, homomorphic encryption and multi-party computation stand out. Both of these techniques share the characteristic of being able to process data while preserving privacy. Homomorphic encryption allows public servers to perform arithmetic operations on encrypted data without the need for decryption. On the other hand, multi-party computation entails data owners collaborating to compute functions on their collective data without sharing any individual information with each other.

Homomorphic encryption minimizes the communication between the client and the server, resulting in a small burden on the client. However, the latency tends to be high due to the significant computation load on the server. Conversely, in the case of multi-party computation, each participant engages in relatively smaller computations, but the round of communications and the amount of data exchanged are generally larger.

To reconcile these conflicting characteristics, prior research has proposed hybrid methods that leverage both HE and MPC, exploiting the strengths of each approach. By implementing such privacy-preserving ML systems, it becomes possible to achieve practical latency and communication levels, striking a balance between efficiency and privacy preservation.

As the importance of privacy-preserving machine learning dealing with encrypted data is emphasized, many researchers are conducting research related to this in various ways. Unlike the existing data itself, when someone wants to use machine learning for sensitive data (medical data, personal information, etc.), it has a lot of problems. Recently, fully homomorphic encryption and MPC are used for PPML. Figure 2.5 explains the procedure of privacy-preserving machine learning.

When it comes to privacy-preserving machine learning, it is widely recognized that multi-party computation is generally less suitable compared to fully homomorphic encryption. This is primarily because most MPC protocols are based on secret-sharing (SS) and are better suited for integer-based computations rather than machine learning tasks for complex number.

Nevertheless, researchers are actively exploring privacy-preserving algorithms using multi-party computations from various perspectives. One notable framework is SecureML [16], which is a deep learning framework designed to utilize additive secret sharing and garbled circuits. Similar to Cryptonets [17], SecureML employs polynomial approximations to simulate non-linear activation functions. Similarly, MiniONN [18] is based on additive secret sharing and Garbled Circuit(GC) but offers lower latency. ABY3 [19], on the other hand, is specifically designed for three-party computation (3PC) and incorporates polynomial approximations for activation functions in neural networks. DeepSecure [20] is an algorithm that reduces the number of non-XOR gates required for privacy-preserving deep learning models using Garbled Circuit. Notably, the neural network model used in DeepSecure has the advantage of being a general convolutional neural network (CNN).

In addition to the multi-party computation protocols mentioned above, there are also studies on information-hiding machine learning using SPDZ [14], which is one of the most well-known multi-party computation protocols. SPDZ stands out as a pro-

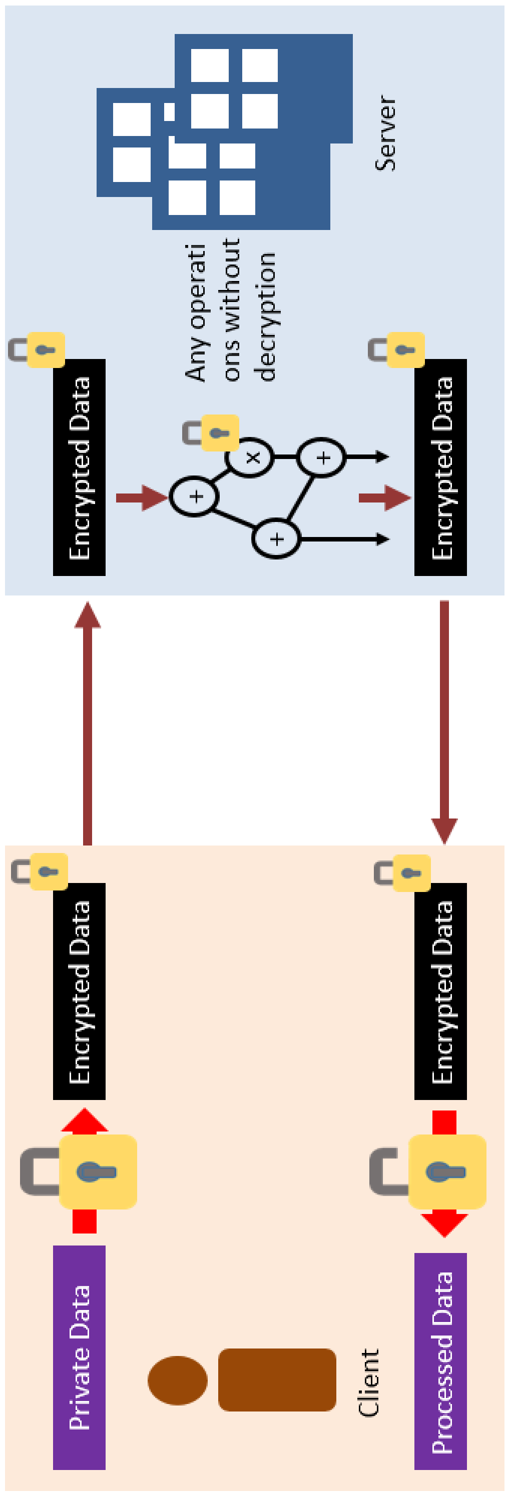


Figure 2.5: Privacy-preserving machine learning.

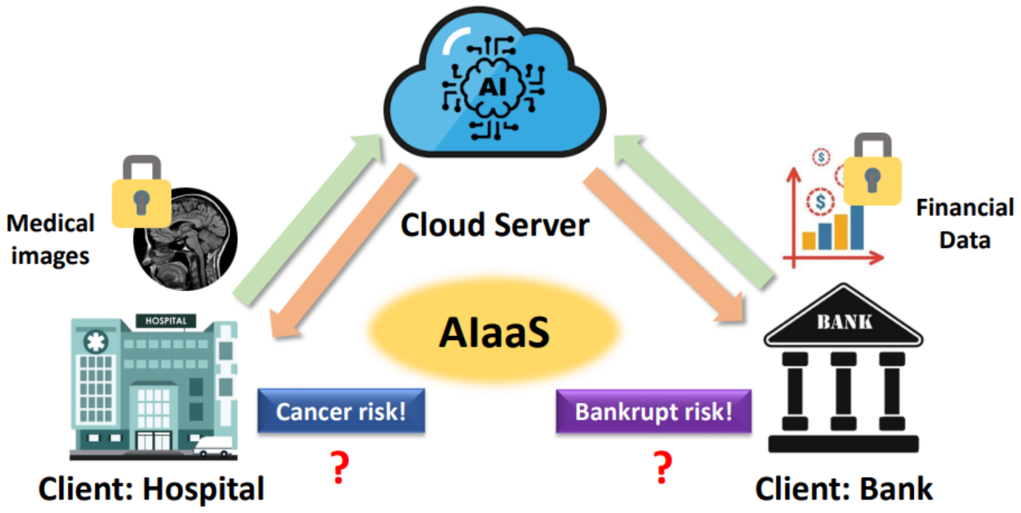


Figure 2.6: Application of Privacy-preserving machine learning.

tocol that can operate seamlessly even when $(n-1)$ out of n users are malicious. It differs from most multi-party computation approaches that focus on two-party or, at most, three-party scenarios, making it distinct from previous studies. Although there may be different interpretations of this direction, it is a field that demands attention from many researchers due to its real-time calculation capabilities and a study suggesting that machine learning performance degradation is not as significant as anticipated, even with an increasing number of users.

Furthermore, there are hybrid schemes that combine the strengths of fully homomorphic encryption and multi-party computations, known as HE + MPC hybrid schemes. One notable method is the client-aided model introduced in nGraph-HE2 [22], which is a representative hybrid approach. It leverages the communication between the server and client to perform operations such as maxpooling and ReLU (non-polynomial functions) on ciphertext within a two-party computation framework. In other words, this method resolves the challenge of using existing multi-party computations in machine learning by utilizing the communication environment. While this study offers the advantage of refreshing ciphertext, addressing one of the major drawbacks of fully homomorphic encryption, it also introduces a potential risk of information leakage about the model during the communication process.

All the privacy-preserving machine learning works mentioned above, which are related to homomorphic encryption, employ HE-friendly networks. These networks are modified convolutional neural networks tailored to accommodate fully homomorphic encryption schemes. However, due to the specific requirements of homomorphic encryption, only shallow network architectures (approximately 3 to 11 layers) are feasible, and only low-degree approximate polynomials can be used for activation functions such as ReLU, ELU, and GeLU. Consequently, these models may not be effective for advanced datasets, and their accuracy tends to be inferior to existing models. Additionally, accessing the entire deep learning framework is necessary, which comes with certain disadvantages. However, their adaptation for homomorphic encryption allows them to be lightweight and fast. However, pre-trained networks offer a significant advantage in that they can utilize ReLU with polynomial approximations and can be applied to well-known datasets. A Comparison between several cases of PPML is shown in Table 2.2.

Table 2.2: Comparison between several cases of PPML

	HE-friendly	FHE	HE+MPC
Accuracy	Low	High	High
Communication cost	Low	Low	High
Security	Satisfy	Satisfy	Satisfy
Model exposure	No	No	Partially yes
Re-training	Need	No	No
Server computation	Low	High	Low

Fully homomorphic encryption only supports addition and multiplication and CKKS only supports one-dimensional data and cyclic shift rotation. To deal with these limitations, the computations necessary to apply the CKKS algorithm to CNN have to be modified. In Table 2.3, operations and complexity in a convolutional neural network with homomorphic encryption are explained.

Table 2.3: Operations and complexity in CNN with HE

CNN	Plaintext	Complexity	Homomorphic encryption	Complexity
Convolution	Matrix multiplication	High	Add/Mult/Rot	Low
Pooling	Maxpooling	Low	Approximation	High
Activation	ReLU	Low	Polynomial approximation	High
Softmax	Softmax	Low	Approximation	High

Convolution is a fundamental operation in convolutional neural networks, where a 3-dimensional tensor serves as both the input and output of the convolution process. In [8], they proposed an efficient convolution method specifically designed for homomorphic encryption. This method involves converting 2-dimensional images and filters into 1-dimensional representations.

More recently, the concept of multiplexed parallel convolution is introduced in [5], which enables even more efficient convolutions to be performed. The CKKS algorithm, commonly used in homomorphic encryption, supports average pooling as a straightforward operation. However, implementing max pooling poses greater challenges within this context. Minimax polynomial was introduced in [2], which can accurately approximate the maximum function through a composition of functions. This technique minimizes the number of operations required within the CKKS scheme.

Activation functions play a crucial role in allowing deep learning models to learn nonlinear boundaries, and they pose particular challenges in privacy-preserving machine learning based on homomorphic encryption. One potential solution is to use the sign function as an activation function. The minimax approximation technique allows for the evaluation of the sign function using only addition and multiplication operations in fully homomorphic encryption scenarios. This approach provides a way to incorporate activation functions while maintaining the privacy-preserving properties of the machine learning model.

2.2 Key Generation and Management Based on Keystore Seed

2.2.1 Key Generation

In this section, basic concepts of secure key generation using keystore seed are given. There is a keystore seed $K = K(0)K(1)\cdots K(L-1)$, which is a randomly generated L-bit binary sequence, where $K(i)$ is the i-th bit of the keystore seed for $0 \leq i \leq L-1$. Users secretly share keystore seed as in Figure 2.6.

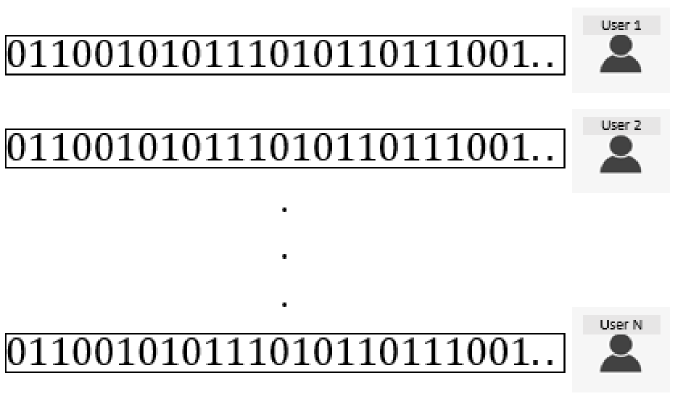


Figure 2.7: Secretly shared keystore seed.

Let a_j be a sub-sequence of length l of the keystore seed and let m_j be a keystore seed index of the first element of a_j, where $0 \leq m_1 < m_2 < \ldots < m_t \leq L-1$. Then, a_j is represented as $a_j = K(m_j)K(m_j+1)\cdots K(m_j+l-1)$. The key k_i of length l is generated as $k_i = a_1 \oplus a_2 \oplus \cdots \oplus a_t$, where $\oplus$ denotes the bit-wise exclusive OR. The set of all possible keys generated from the keystore seed K is denoted as $\Psi = \{k_i | 1 \leq i \leq \Lambda\}$, where Λ is $\binom{L}{t}$. This key generation method is expressed as the (L, l, t)-key generation scheme, where l is the length of each key and t is the number of subkeys of keystore seed for the generation of each key. Table 2.4 shows the number of generated keys with parameters (L, l, t)

Table 2.4: The number of generated keys

$(t,l) = (5,128)$					
L =	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
$\Lambda \geq$	2^{53}	2^{58}	2^{63}	2^{68}	2^{73}
$(t,l) = (10,256)$					
L =	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
$\Lambda \geq$	2^{108}	2^{118}	2^{128}	2^{138}	2^{48}

2.2.2 Key Management

After key generation, the generated keys can be used in the following way:

1) A file is encrypted using a key k_i randomly selected from set Ψ.

2) Attach the key index information $i = (m_1, m_2, \cdots, m_t)$ into the encrypted file and send it.

3) To decrypt an encrypted file, the encryption key k_i is regenerated from the secure stored keystore seed and the received file using k_i is decrypted using the attached key index information i.

The keystore seed should be protected in a secure memory that cannot be accessed by outside users. Even though the key index information is available, any information on the keystore seed should not be disclosed.

2.2.3 Information-Theoretically ϵ-Secure Keystore

The information-theoretically ϵ-secure for arbitrarily small ϵ is defined according to the following specifications.

1) A keystore $\Psi = \{k_i \mid 1 \leq i \leq \Lambda\}$ of keys of length l generated from a keystore seed K is said to be information-theoretically ϵ-secure for $0 \leq \epsilon < 1$, if the properties in the following theorems hold.

2) For $1 \leq i \leq \Lambda$ and arbitrarily small $\epsilon > 0$, all keys k_i are randomly and uniformly distributed over $\{0,1\}^l$ as

$$\Pr\{k_i = k_j\} \leq (1-\epsilon) \times 2^{-l} + \epsilon.$$

3) For all pairs of independent indices $i, j, 1 \leq i, j \leq \Lambda$,

$$H(k_j | i, j, k_i) \geq H(k_j | j) \times (1-\epsilon) = l(1-\epsilon).$$

Yang and Wu [46] stated that the 3) can be extended to the following argument.

Argument 1 (nth order of 3)). *For all independent* $i, j_1, \ldots, j_n$, *where* $1 \leq i, j_1, \ldots, j_n \leq \Lambda$, *have*

$$H(k_i | j_1, \ldots, j_n, i, k_{j_1}, \ldots, k_{j_n}) \geq \\ H(k_i | i) \times (1-\epsilon) = l(1-\epsilon). \tag{2.1}$$

They insist that by utilizing the information-theoretically secure keystore Ψ generated from the shared keystore seed K, key distribution becomes both secure and convenient. To encrypt a file, one randomly selects a key, denoted as k_i, from the keystore. This k_i is then used as the encryption key to encrypt the file using a symmetric cipher like AES. Subsequently, the key index i is inserted into the ciphertext header. The encrypted file is composed of the key index i and the ciphertext itself. When a legitimate recipient receives the encrypted file, they retrieve the corresponding key k_i from K using the key index i, and utilize k_i to decrypt the ciphertext.

Distributing k_i through i does not reveal any information about the key k_i itself, as the mutual information between k_i and its index i is zero. Moreover, since k_i can be easily derived from K and i, there is no need to physically store Ψ. As a result, the challenge of managing a large list of random keys k_i is essentially reduced to managing a single keystore seed K. Consequently, the burden of key management for long-term data protection is significantly mitigated.

They have successfully tackled the challenges associated with key generation and management to enable secure encryption with a one-key-for-one-file approach, ensuring that each file is encrypted using a unique key. Their approach considers the information-theoretic aspect of security and introduces the concept of information-theoretical security to evaluate the security of a keystore generated from a random string of L bits. They have proposed an efficient keystore generation scheme, resulting in an information-theoretically secure keystore Ψ with a small value. The combination of the information-theoretically secure Ψ, the large number of keys, and the ease of generating each key k_i from the keystore seed and key index address the major challenges in distributing and managing a substantial number of random keys for long-term data protection in public cloud environments.

Chapter 3

Client-Aided Deep Neural Network on Fully Homomorphic Encryption without Bootstrapping in the Client-Server Model

3.1 Introduction

Research has primarily focused on the theoretical exploration of technologies that enable operations while preserving the secrecy of a user's input. This emphasis arises from previous encryption methods that prioritized corporate security over safeguarding personal information. However, recent studies have spurred users to pay more attention to their personal data, leading to increased interest in research on anonymization techniques such as homomorphic encryption and multi-party computations. Machine learning has emerged as a central component of contemporary research and finds extensive application across various domains. As previously discussed, the growing significance of personal information underscores the need to emphasize privacy preservation in machine learning. Privacy-preserving machine learning, commonly referred to as PPML, has become a highly relevant subject in the fields of security and cryptography, undergoing rapid investigation in recent years.

Machine learning has undergone rapid development in recent years and currently outperforms human intelligence in specific domains. As recent ML models demonstrate remarkable performance improvements, they are being rapidly deployed across many industries. In this context, the privacy issue represents a crucial challenge that artificial intelligence must address. With the rise of significant legal concerns regarding privacy in ML services involving governments, enterprises, and clients, the societal value of privacy has gained increasing importance. Efficient implementation of ML systems that ensure the protection of user data privacy can enable clients to utilize these ML services without worrying about data leakage, thereby fostering growth in the artificial intelligence (AI) industry.

Two practical cryptographic tools used to design privacy-preserving AI systems are homomorphic encryption and multi-party computation. Both approaches share the characteristic of processing data while preserving privacy. HE refers to an encryption scheme designed to allow public servers to perform arithmetic operations on encrypted data without decryption. On the other hand, MPC involves a protocol in which data owners communicate with each other to compute functions on the entire dataset without sharing any information about their individual data. HE requires minimal communication between the client and the server, resulting in a smaller burden on the client but higher latency due to extensive computation on the server. However, MPC involves relatively small computation for each participant but typically requires multiple rounds of communication with larger amounts of data exchanged. To reconcile these conflicting characteristics, prior research has developed hybrid methods that leverage the advantages of both tools, resulting in the implementation of practical privacy-preserving machine learning systems with manageable latency and communication requirements.

In this chapter, examples of privacy-preserving machine learning using simple fully homomorphic encryption and multi-party computation and the most recent research are introduced and compared. In Section 3.2, PPML frameworks using FHE, especially the 2022 ICML paper using pre-trained parameters, are introduced and their pros and cons are analyzed [5]. In addition, PPML algorithms using MPC and DEL-

PHI, the most recent result and attention-grabbing algorithm, are introduced and their features are analyzed. Finally, in Sections 3.3 and 3.4, the main contribution of this dissertation, the bootstrapping replacement method for PPML on FHE in the communication environment, is introduced, and what advantages it has over the previously introduced algorithms and experimental results are introduced.

3.2 Cryptographic Inference on Deep Neural Networks

3.2.1 Inference of Deep Convolutional Neural Networks on Fully Homomorphic Encryption

Currently, there is a significant amount of attention being directed toward a study that aims to perform image classification using the CKKS algorithm, which is widely recognized as the most popular algorithm in the field of homomorphic cryptography. The study, published in the ICML 2022 [5], not only addresses the challenges associated with CKKS in machine learning but also achieves the remarkable feat of performing inference on CIFAR-10/100 images using ResNet models with pre-trained parameters, marking the first instance of such an accomplishment. In this section, I will introduce the paper that can be served as the foundation for my own study.

The utilization of pre-trained networks holds paramount importance in privacy-preserving machine learning studies employing homomorphic encryption. As previously mentioned, PPML using homomorphic encryption has predominantly relied on HE-friendly networks due to the inherent difficulty in implementing the ReLU activation function. However, the work in [5] pioneers the use of a pre-trained network within the context of homomorphic encryption and introduces two key techniques.

The first technique is known as multiplexed parallel convolution, which involves compacting the existing three-dimensional tensor into a one-dimensional vector, thereby enabling the design of one bootstrapping operation per layer as in Figure 3.1. This innovative approach yields a remarkable performance improvement, surpassing previous studies by a factor of 134.

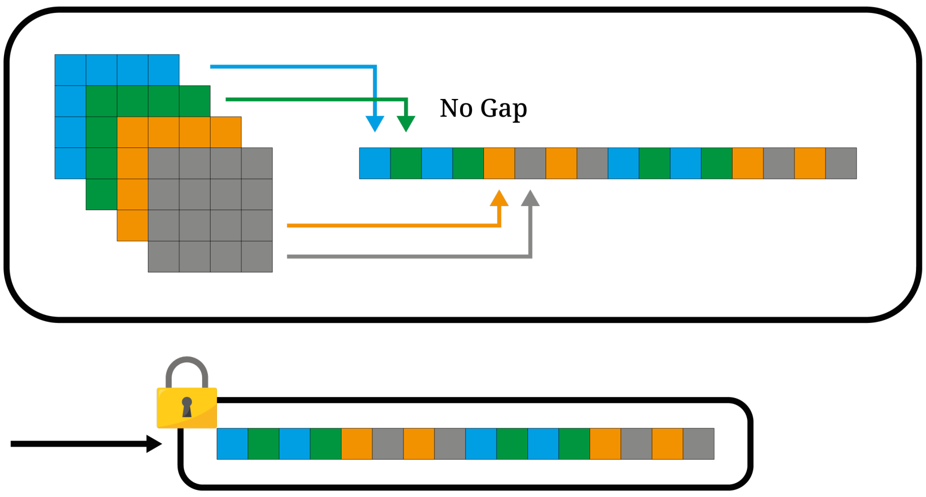


Figure 3.1: Multiplexed parallel convolution.

The second technique is known as imaginary-removing bootstrapping in Figure 3.2. When performing deep neural network using fully homomorphic encryption, a phenomenon known as catastrophic divergence can occur due to small imaginary errors, particularly in the case of ReLU activation. In fact, there is a 25% probability of encountering catastrophic divergence in a single layer. However, the introduction of the imaginary-removed bootstrapping technique effectively mitigates these situations, ensuring the reliability and stability of the computations.

In Figure 3.3, you can see how the RNS-CKKS scheme was implemented for each ResNet model [5]. My research is also conducted in the same way, but only the bootstrapping part is replaced with ciphertext refresh using communication between client and server. It is crucial because the amount of computation for bootstrapping is very large.

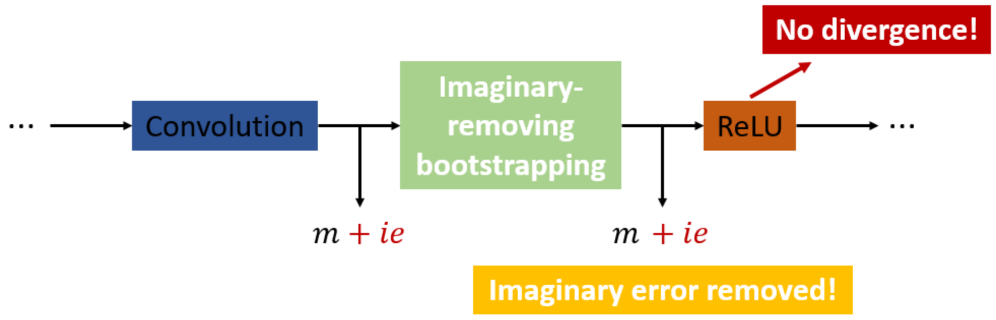


Figure 3.2: Imaginary removing bootstrapping.

3.2.2 Inference on Multi-Party Computation of Deep Convolutional Neural Networks

DELPHI, which was introduced in 2020, presents an exciting advancement in cryptographic neural network inference through the utilization of secure multi-party computation [7]. Prior to the publication of this groundbreaking work, GAZELLE [8] had emerged as the representative approach for MPC-based privacy-preserving machine learning. In GAZELLE, linear operations were conducted using homomorphic encryption, while non-linear operations, such as the ReLU, were performed using garbled circuits. DELPHI, however, revolutionizes the performance of GAZELLE by introducing novel enhancements for ReLU.

One notable improvement in DELPHI involves the incorporation of preprocessing techniques aimed at reducing the actual inference time. This addition proves instrumental in expediting the overall computation. Moreover, instead of relying solely on homomorphic operations, DELPHI introduces MPC based on secret sharing for the linear step. By doing so, it effectively departs from the sole reliance on homomorphic operations, thus broadening the range of cryptographic techniques employed. Another significant modification is observed in the treatment of ReLU computations. Unlike the previous approach, which solely utilized garbled circuits, DELPHI combines garbled circuits with beaver triples. Although the use of beaver triples enables faster opera-

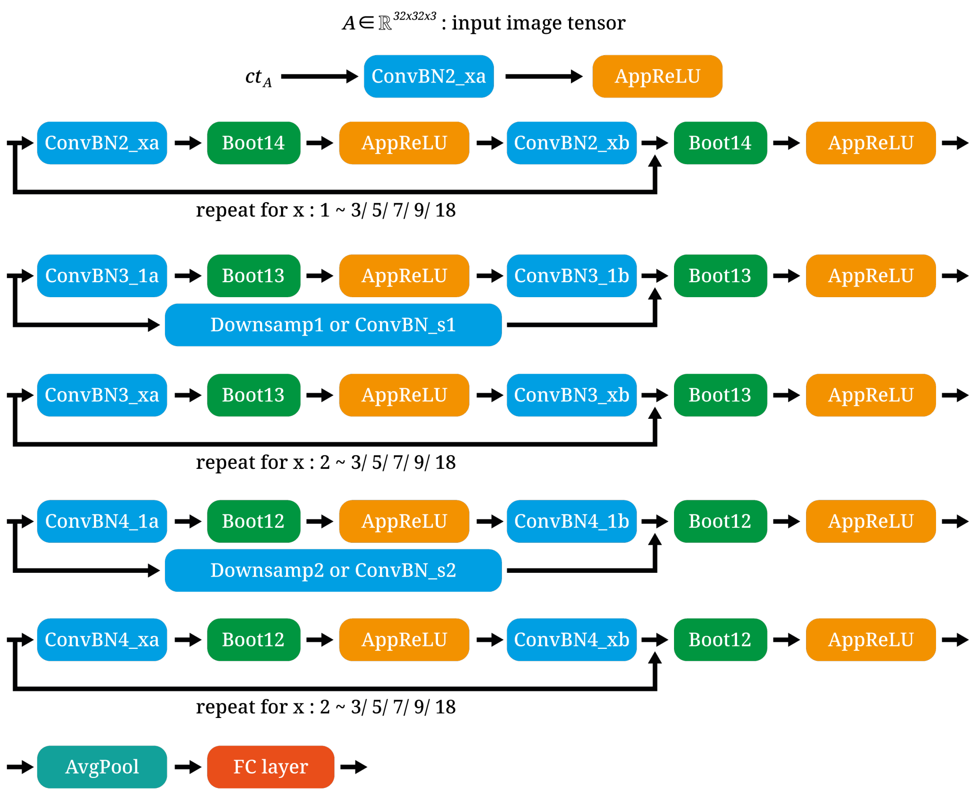


Figure 3.3: Structure of the proposed ResNet on the RNS-CKKS scheme.

tions, it presents a challenge in terms of inference accuracy. On the other hand, by utilizing garbled circuits with exact ReLU computations, DELPHI ensures a sufficiently high level of accuracy. However, this approach comes at the cost of a significant increase in the amount of data to be transmitted, leading to a corresponding increase in running time.

DELPHI's contributions extend beyond mere performance improvements. By utilizing a combination of garbled circuits, beaver triples, and parallel processing, DELPHI effectively limits the extent of homomorphic operations in the offline secret sharing phase. This optimization substantially enhances performance while preserving the privacy and security of the computation. Notably, the online phase solely consists of straightforward calculations, resulting in minimal computational overhead. In terms of communication requirements, the amount of data transmitted during preprocessing(offline phase) and online processing in GAZELLE remains comparable. However, DELPHI increases the overall data size and transmission speed for the linear portion of the computation. Nevertheless, by strategically dividing the process into online and offline phases, DELPHI minimizes the costs associated with client services. In contrast, GAZELLE does not employ preprocessing for linear operations but instead utilizes it exclusively for garbled circuits. Considering these factors collectively, DELPHI roughly doubles the computational performance of GAZELLE.

Another noteworthy contribution of DELPHI lies in its ability to expedite computations through the combined utilization of garbled circuits, beaver triples, and garbled circuits for non-linear operations. However, to mitigate the decrease in accuracy resulting from the inclusion of beaver triples, DELPHI leverages an optimized selection of layers that employ both garbled circuits and beaver triples. This process employs a type of neural architecture search (NAS) algorithm, which automates the search for the optimal neural network architecture that satisfies specific conditions. NAS typically involves training multiple neural networks, evaluating their accuracies, and ultimately selecting the one that demonstrates superior performance.

In DELPHI's offline phase, which employs homomorphic encryption, the client possesses the value $M_i r_i - s_i$, while the server holds s_i. Combining these values enables the derivation of $M_i r_i$, although both parties remain in a secret-sharing state, unaware of the actual answer. For non-linear operations, two sets of values are shared. One set involves sharing parameters for garbled circuits, while the other set is shared using oblivious transfer for beaver triples. Linear operations are subsequently performed using the previously shared secret values.

The computation commences with the client calculating $x - r_1$ using the offline-generated value r_1 and its confidential input x. This result is then transmitted to the server, which performs a single linear operation using $x - r_1$ and concludes the linear step. For each subsequent layer, the client possesses an offline-generated value r_i, while the server is aware of the output value x_i of the i-th layer, subtracted by r_i. The intermediate output x_i remains unknown to both the server and the client. At this stage, the server performs the calculation $M_i(x_i - r_i) + s_i$ using M_i and s_i, subsequently sending the garbled circuit label of this value to the client. The client, utilizing the values obtained through offline oblivious transfer, calculates and acquires a one-time pad ciphertext $x_{i+1} - r_{i+1}$. This value is then transmitted back to the server, which can recover $x_{i+1} - r_{i+1}$. As the operations progress through the final layer, the server forwards the ultimate value $x_l - r_l$ to the client, who can then retrieve the final output x_l.

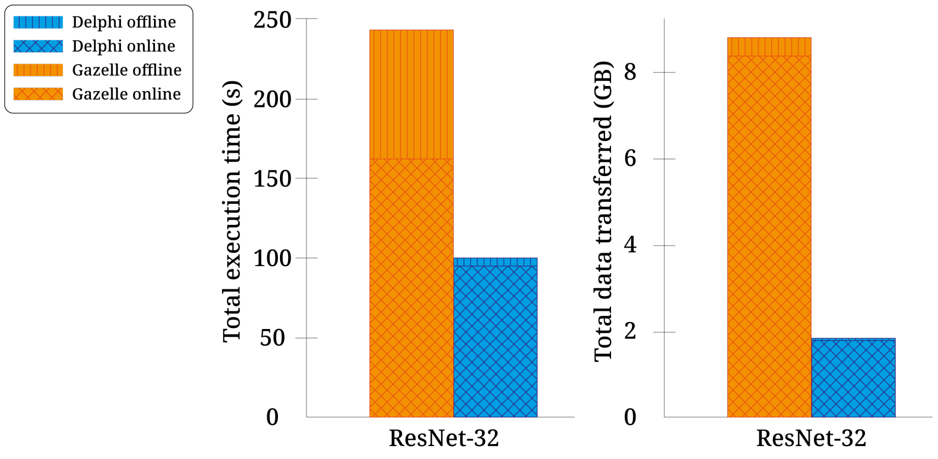


Figure 3.4: DELPHI simulation results.

Figure 3.4 shows the simulation result of DELPHI and GAZELLE. For one CIFAR-100 image using Resnet-32, it cost 250s and over 8GB when using exact ReLU which is the same as GAZELLE's result. DELPHI claims that their best case could reduce execution time to 100s and communication cost to under 2GB.

3.3 Client-Aided Inference Using Communication Cost

3.3.1 Replacing Bootstrapping Using Communication Cost

GAZELLE and DELPHI are prominent hybrid models that combine homomorphic encryption and multi-party computation. However, these models still exhibit a significant amount of communication, which limits their practical use. In the case of the DELPHI model, when performing the ResNet-32 model, both online and offline stages require over 8GB of communication per image. This substantial communication overhead places a burden on clients and hinders the smooth utilization of AI services. Although these models have focused on minimizing online communication and latency when processing selected images, the overall resource consumption remains high, as clients need to continuously utilize substantial communication and computational resources. Therefore, it is essential to reduce communication overhead while implementing a privacy-preserving machine learning system based on the hybrid approach.

The primary reason for the large communication overhead is the utilization of the garbled circuit technique for performing ReLU operations, which is a form of multi-party computation. Previous studies have regarded homomorphic encryption as suitable for linear operations, while multi-party computation is considered appropriate for non-linear operations on a bit-wise level. Consequently, in privacy-preserving convolutional neural network models, convolution operations are typically processed using homomorphic encryption, while the ReLU activation functions are implemented using garbled circuits. However, due to the large number of ReLU functions to be computed and the substantial communication required for each ReLU function, the ReLU functions using garbled circuits account for a significant portion of the overall communication volume, playing a critical role in the extensive communication requirements. To mitigate the communication overhead, it may be desirable to explore new hybrid PPML models that do not rely on garbled circuits for ReLU functions. Previous work of HE-based privacy-preserving machine learning has the disadvantage of bootstrapping, which takes a very huge amount of total execution time. Bootstrapping should be applied in order to perform an arbitrary number of operations in fully homomorphic encryption. In particular, in order to execute image inference in a state of FHE that is applied to deep neural networks like ResNet, bootstrapping is needed thousands of times. This one bootstrapping takes about 300 seconds, although it depends on the system in the currently implemented library. For this reason, it takes more than 3 hours to classify a simple image such as CIFAR-10 in ResNet-20. Thus reducing the number and time of bootstrapping is important to improve the performance of privacy-preserving machine learning using fully homomorphic encryption.

Recent work introduced another new framework for privacy-preserving machine learning models. These models have a different purpose compared to hybrid PPML models, as they assume that the client does not actively participate in the AI operations. Instead, they solely rely on fully homomorphic encryption, which allows unlimited addition and multiplication operations on encrypted data. It is important to note that GAZELLE and DELPHI use a partially homomorphic encryption scheme, which does not support multiplication between encrypted data. However, the implementation by [5]. relies solely on homomorphic encryption without leveraging multi-party computation techniques, resulting in significant latency. While this implementation minimizes the clients' resource usage, if clients can participate in the computation, reducing latency through MPC techniques would be desirable. To achieve this, further optimization is needed, combining the composite approximation method with MPC techniques.

There are several reasons for the significant latency in [5]. Firstly, the bootstrapping operation, which elevates the ciphertext level from zero to enable further homomorphic operations, incurs substantial latency compared to other homomorphic operations. If clients' involvement in operations is allowed to some extent, replacing the bootstrapping operation with MPC techniques is recommended. Another factor contributing to the latency is the requirement for a large depth in the implementation. For sufficient security in fully homomorphic encryption schemes, there is an upper limit on the total bit-length of the modulus, determined by the product of the evaluation modulus and the special modulus. A larger ratio between the evaluation modulus and the special modulus results in increased runtime for the key-switching operation, which is a fundamental operation in many homomorphic operations. By reducing this ratio while maintaining the same parameters, the overall runtime of the operations can be significantly decreased.

In this dissertation, I focus on the feasibility of an algorithm that can replace bootstrapping without leaking each other's information using multi-party computations in a situation where a communication environment is available. However, the server should not give information about its artificial neural network model to the client and also the server should not gain the client's personal data. This method is based on information theoretically safe so that complete information protection is possible. A random number is added to the message requiring bootstrapping and then transmitted to the client. After the client decrypts the message, re-encrypt with a large modulus ciphertext. Results are sent back to the server. This algorithm is summarized as follows :

1) Random Addition
   Server execute $Enc_o(m) + Enc_o(r) -> Enc_o(m + r)$
   Send to a client to refresh

2) Ciphertext Refresh
   Client execute $Enc_o(m + r) -> (m + r) -> Enc_L(m + r)$
   Send to server

3) Random Deletion
   Server execute $Enc_L(m + r) + Enc_L(-r) -> Enc_L(m)$
   Get refrehsed ciphertext

More details of replacing bootstrapping using communication can be found in Figure 3.5.

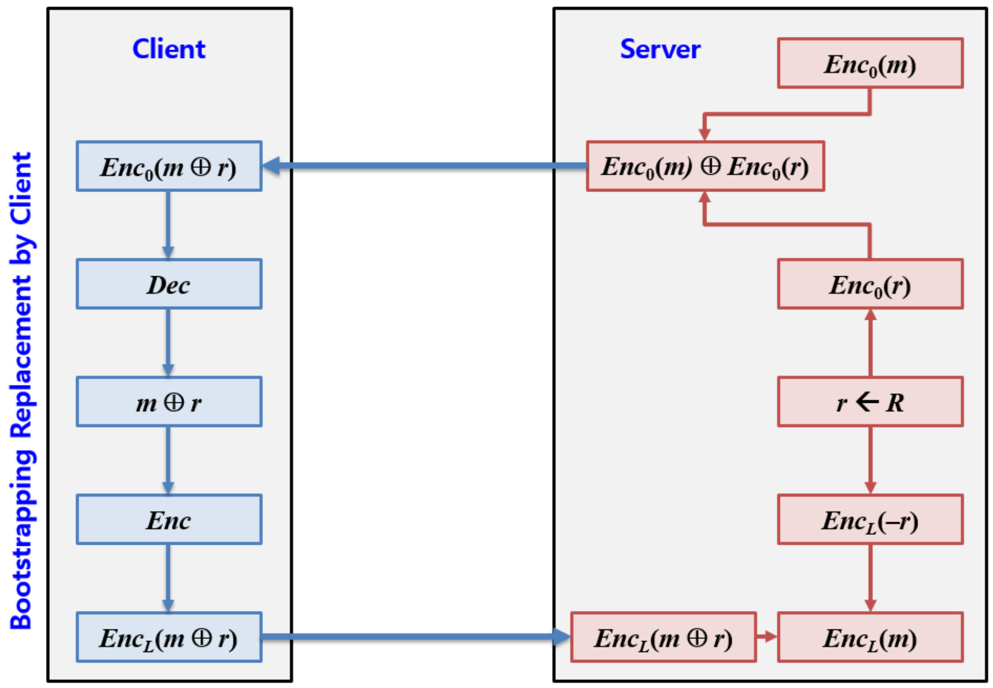


Figure 3.5: Bootstrapping replacement by the client in the PPML on FHE.

To reduce communication, I eliminate the use of garbled circuits, which require significant communication, and instead employ the composite approximation method for computing ReLU functions. This allows all ReLU operations to be performed on the server using homomorphic encryption. The composite approximation method, previously utilized in HE-based PPML models by [5]., has been proven to achieve the required accuracy for computing ReLU functions in privacy-preserving AI systems.

In GAZELLE and DELPHI, which only handle the convolution operation using homomorphic encryption, a linear HE scheme is used, supporting linear operations but not multiplication between ciphertexts. However, in the composite approximation method, multiplication between ciphertexts is necessary for polynomial evaluation. To facilitate this method, I adopt the CKKS fully homomorphic encryption scheme, which supports multiplication between ciphertexts in addition to linear operations. Additionally, I propose the mod-raised one-time pad (MR-OTP) technique to replace the bootstrapping operation in HE. The bootstrapping operation in CKKS enables additional operations by converting a lower-level ciphertext into a higher-level ciphertext while preserving the encrypted data. MR-OTP serves the same purpose as bootstrapping in [5]. While the one-time pad technique has been widely used in other MPC techniques, I require a more sensitive analysis in this new OTP setting to ensure negligible decryption failure, as my objective is to raise the ciphertext level, equivalent to enlarging the domain of the ciphertext.

However, a simple replacement of bootstrapping operations with MR-OTP is not sufficient, as it still results in considerable latency compared to the DELPHI model. Therefore, I optimize the modulus chain to reduce elementary homomorphic operations. Since bootstrapping operations are not employed in my implementation, the evaluation modulus can be halved compared to the approach of [5]. By analyzing the runtime of key-switching operations for various ratios and optimizing the modulus chain, I minimize overall latency. Thus, I can optimize the ratio between the bit length of the evaluation modulus and that of the special modulus.

Algorithm 3.1 shows how to refresh the ciphertext with generated random values. In this process, additional considerations are needed to generate random values. It is used for masking the message, but if an excessively large value occurs, $m + r$ may be distorted by mod q. To do this, fine adjustment is required for the value corresponding to r, and the routine for processing the corresponding value is shown below. A random value that has been selected between $(-q_0/(2 * ctxt.scale()) + 1)$ to $(q_0/(2*ctxt.scale())-1)$, has a failure probability and it has to be lower than 2^{-128} to satisfy 128-bits security. In this case for parameter $N = 2^{16}$, the probability is $1-NP$ and $P = 2/2^{200}$, able to confirm that a random value is selected safely enough.

Algorithm 3.1 Ciphertext Refresh Algorithm

```
1: **Step 1) Random generation**
   Set $abc$ as $q_0/(2 * ctxt.scale())$ .
   Randomly pick $m_b$ from the range of $(-q_0/(2 * ctxt.scale()) + 1)$ to $(q_0/(2 * ctxt.scale()) - 1)$
2: **Step 2) Encrypt the random value**
   First encode the random as $encoder.encode(m\_b, ctxt.scale(), plain\_b)$ and then encrypt $encryptor.encrypt(plain\_b, cipher\_b)$ $cipher\_f = cipher\_b$
3: **Step 3) Added to low level ciphertext ctxt refresh it**
   $evaluator.mod\_switch\_to\_inplace(cipher\_b, ctxt.parms\_id())$
   $evaluator.add(cipher\_b, ctxt, cipher\_c)$ decrypt and re-encrypt it to refresh
   $decryptor.decrypt(cipher\_c, plain\_c)$
   $encoder.decode(plain\_c, output\_c)$
   $encoder.encode(output\_c, ctxt.scale(), plain\_c)$
   $encryptor.encrypt(plain\_c, cipher\_d)$
   $evaluator.mod\_switch\_to\_inplace(cipher\_f, cipher\_d.parms\_id())$
4: **Step 4) Finally, subtract the added random value to get the refreshed ciphertext rtn**
   $evaluator.sub(cipher\_d, cipher\_f, rtn)$
```

3.3.2 Level Consumption of Proposed Method

When executing the above procedure, it is important to consider the level of consumption in each process. In bootstrapping, more than 10 levels are consumed, but in the proposed method, there is no level consumption. As a result, it becomes possible to perform more operations than when using bootstrapping. However, optimization is necessary in this case. Once the low-level ciphertext is processed, its size is reduced, and operations can be performed faster compared to high-level ciphertext. If there are many computations for a convolutional neural network at a specific level, more time is required for the same computation if it is performed at a high level. Therefore, even if the level for bootstrapping is saved, using it as the level of the CNN unconditionally does not guarantee good performance.

Furthermore, in the past, the same level of consumption was used for each layer of the CNN, but it is necessary to optimize the operation between odd and even layers through the added level. Similarly, since the size of the ciphertext varies according to the level, and this difference directly affects the operation speed, the algorithm should be improved to speed up the overall operation.

I propose two types of ciphertext refresh using different levels as follows :

- Proposed 1) : Using level 18, which uses one layer for one refresh
- Proposed 2) : Using level 35, which uses two layers for one refresh

Unlike the previous work that performs bootstrapping when the level goes to 0, this work performs ciphertext refresh when the level is one. Figures 3.6 and 3.7 show the difference in level consumption between the two methods. In previous work, the level consumption for ReLU is 15, the convolution is 2, and bootstrapping consumes 14. Therefore, the previous method proceeds as 31-0-17-0-17-0... meaning bootstrapping modifies at level 31, and after performing slottocoeff, modular reduction, and coefftoslot, it becomes level 17. Proposed 1) executes 18-1-18-1-18-1..., and Proposed 2) executes 35-1-35-1-35... Both proposed methods have their advantages. Algorithm

3.2 shows the procedure of level adjustment for two proposed methods.

Firstly, they have almost the same communication cost because Proposed 1) doubles the number of communications, but half the communication amount. More specifically, the communication in my work depends only on two ciphertext sizes: high-level ciphertext size(after refreshing at the client) and low-level ciphertext size(before refreshing at the server). RNS-CKKS's ciphertext size is approximately $N \times Level \times 16$, and thus the communication cost of Proposed 1) is $layer \times 2 \times N \times 16 \times (18 + 1)$ and Proposed 2)'s communication cost is $layer \times N \times 16 \times (35 + 1)$. Proposed 2) has a slightly smaller communication cost by $2 \times layer \times N \times 16$ compared to that of Proposed 1).

Proposed 1) has an advantage in bandwidth-limited situations. Proposed 2) has an advantage in terms of the disadvantages of multi-party computation. MPC has a flaw in that all users (or participants) have to be online during communication. Proposed 1) has twice the number of communications, requiring the client to be online twice compared to Proposed 2). Nevertheless, since Proposed 1) has a smaller high level ciphertext compared to Proposed 2), I focus on Proposed 1), and all the experimental results to be introduced next are carried out based on Proposed 1). After the server obtains refreshed ciphertext, it can control the level. This has a significant impact on simulation speeds. For example, when using level 30 to infer one CIFAR-10 image in ResNet-20, it takes 1,551 seconds. However, by reducing the level from 30 to 18, which is the minimum level to handle one layer, the simulation time reduces to 722 seconds. Lowering the high level from 30 to 18, I am able to achieve the fastest inference result of 526 seconds together with further optimization.

Algorithm 3.2 Level Adjustment Algorithm

```
1: Step 1) Get refreshed ciphertext rtn
   $evaluator.sub(cipher\_d, cipher\_f, rtn)$
2: Step 2) Change the ciphertext scale
   $size\_tcur\_level = rtn.coeff\_modulus\_size()\ doublescale\_change =$
   $(pow(2.0, 46)/rtn.scale()) * ((double)modulus[cur\_level - 1].value())$
   $encoder.encode(1, scale\_change, scaler)$
   $evaluator.mod\_switch\_to\_inplace(scaler, rtn.parms\_id())$
   $evaluator.multiply\_plain\_inplace(rtn, scaler)$
   $evaluator.rescale\_to\_next\_inplace(rtn)$
   $rtn.scale() = pow(2.0, 46)$
3: Step 3) Lower the level and repeat the below formula
   $evaluator.mod\_switch\_to\_next\_inplace(rtn)$
```

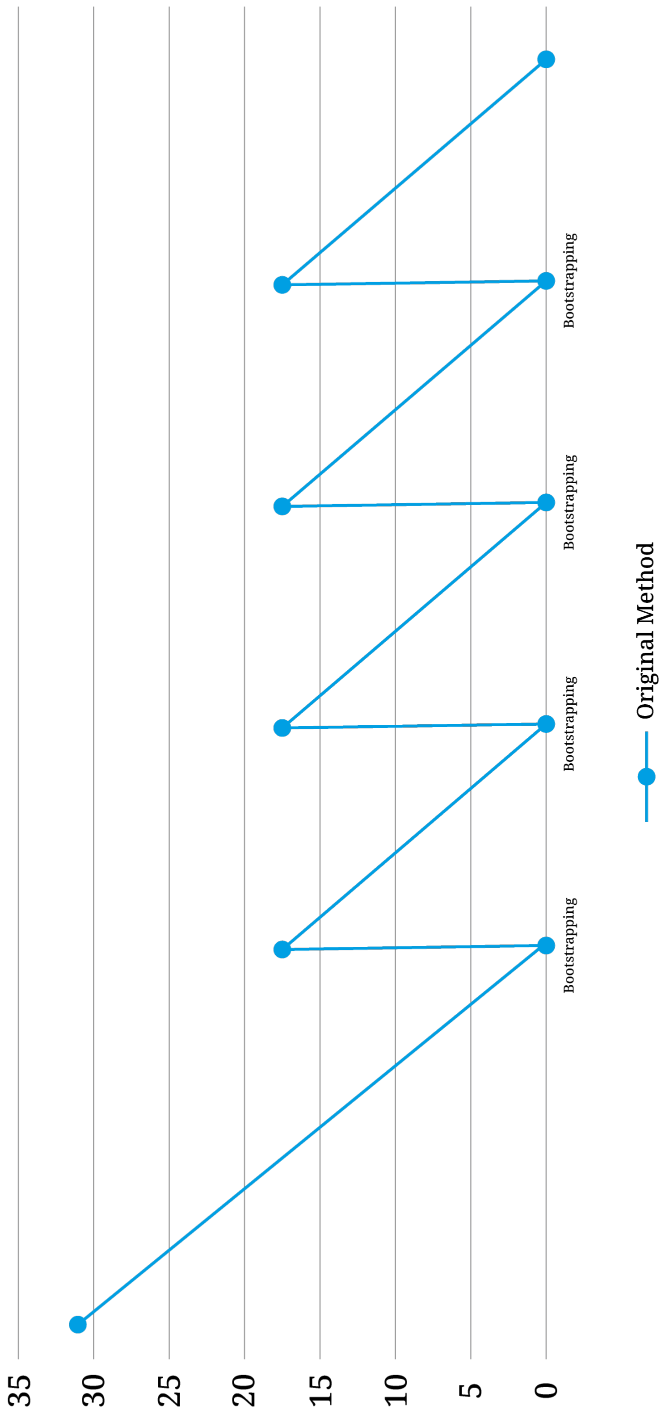


Figure 3.6: Previous method level consumption.

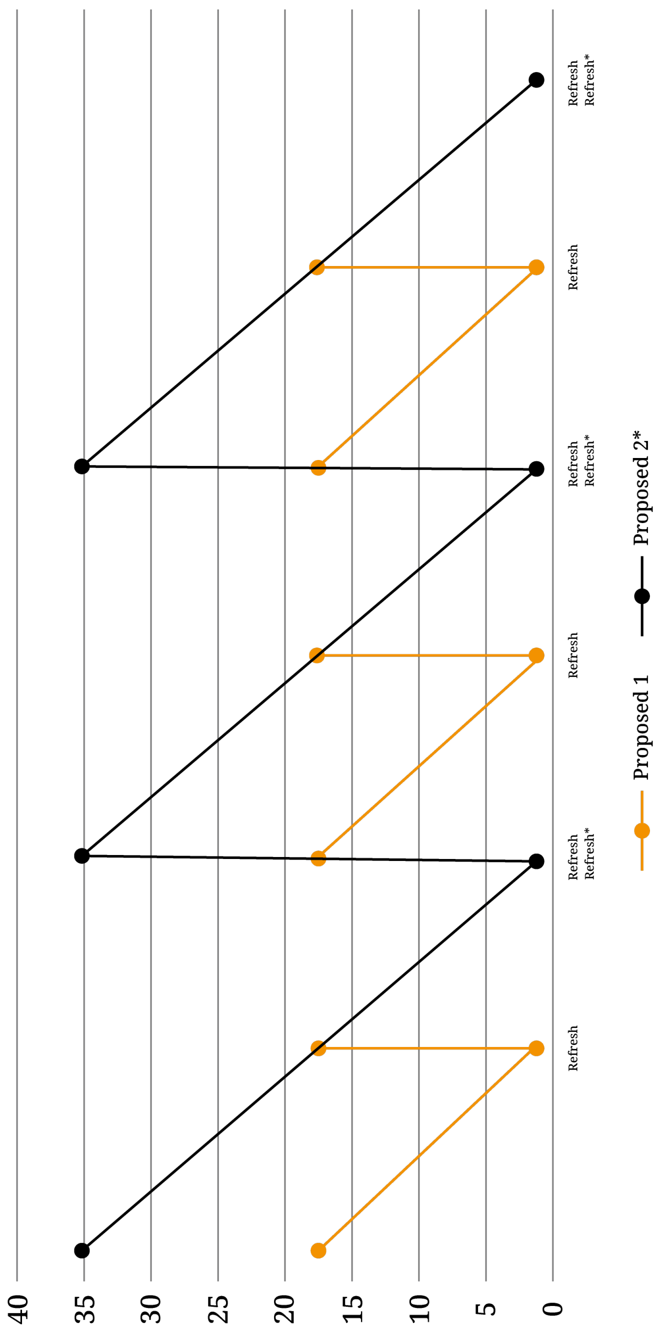


Figure 3.7: Proposed method level consumption.

3.4 Simulation Results

In this section, I perform Proposed 1) for ResNets using datasets CIFAR-10 and CIFAR-100. Both datasets have 50,000 images for training and I use 1,000 images for inference [6]. Using library SEAL [10] on AMD Ryzen threadripper PRO 3995WX with 512 GB RAM, running Ubuntu 20.04 system. Pre-trained parameters are used for standard ResNet-20/32/44/56/110. I set the polynomial degree $N = 2^{16}$ and the full slots $n = 2^{15}$. Starting from level 18 when the level goes to 1, do ciphertext refresh. The other parameters are used in the same as [5] and are described in detail.

3.4.1 Classification Runtime and Communication Cost

Table 3.1 below summarizes how much inference time it takes for one image for each response model. For ResNet-32, CIFAR-100 was also added. It is confirmed that the time reduction is about 4-5 times compared to the previous work [5]. It is confirmed that the number of layers is larger depending on the ResNet model, and thus it has a larger increase in time reduction.

Table 3.1: Classification runtime for one image using ResNet on RNS-CKKS

Image	Model	Previous	Proposed
CIFAR-10	Resnet-20	2,271s	526s
	Resnet-32	3,730s	850s
	Resnet-44	5,224s	1,167s
	Resnet-56	6,852s	1,480s
	Resnet-110	13,282s	2,898s
CIFAR-100	Resnet-32	3,942s	887s

Obviously, communication cost between client and server increases as the number of bootstrapping replacements increases. As previously mentioned, the size of RNS-CKKS's ciphertext is $N * Level * 16$ bytes and thus the communication cost for one image inference could be calculated as the number of bootstrapping * (low level ciphertext size + high level ciphertext size). For example, ResNet-32 has 30 times bootstrapping, and thus 30*65536*(1+18)*16 is 597,688,320, which means the communication cost is 597.688MB. The communication cost for different ResNet models is summarized in Table 3.2.

Table 3.2: Communication cost for one image inference in ResNet

Model	Communication cost
Resnet-20	358.613MB
Resnet-32	597.688MB
Resnet-44	836.763MB
Resnet-56	1075.838MB
Resnet-110	2151.677MB

When using multiplexed parallel convolution, it is possible to implement multiple images in parallel using multiple threads, unlike existing research that can infer only one image at a time. Table 3.3 is the result of measuring the amortized runtime by inferences of 50 images at the same time.

Table 3.3: Classification runtime for multiple images using ResNet on RNS-CKKS

Image	Model	Runtime	Armortized runtime
CIFAR-10(50 images)	Resnet-20	845s	17s
	Resnet-32	1,357s	27s
	Resnet-44	1,937s	39s
	Resnet-56	2,511s	50s
	Resnet-110	4,880s	97s
CIFAR-100(50 images)	Resnet-32	1,439s	29s

3.4.2 Accuracy

Table 3.4 present the accuracy of image classification for CIFAR-10/100 images using ResNet on RNS-CKKS. According to the simulation result, using communication cost to ciphertext refresh does not affect the accuracy at all, which is very close to baseline accuracy and previous work's accuracy [5]. This means that the proposed method has been improved by reducing the computation complexity while maintaining the previous accuracy.

Table 3.4: Classification accuracy of CIFAR-10 and CIFAR-100 images using ResNet model with RNS-CKKS

Dataset	Model	Baseline accuracy	ICML accuracy	Proposed accuracy
CIFAR-10	Resnet-20	91.52%	91.31%	91.3%
	Resnet-32	92.49%	92.4 %	92.7%
	Resnet-44	92.76%	92.65%	92.9%
	Resnet-56	93.27%	93.07%	93.0%
	Resnet-110	93.5%	92.95%	93.8%
CIFAR-100	Resnet-32	69.5%	69.43%	69.6%

3.4.3 Compare to Previous Works

In this section, I compare my simulation results from two perspectives. One is to compare with fully homomorphic encryption inferences introduced in Section 3.2.1, which has the disadvantage that bootstrapping is slow. Another is to compare with multi-party computation inferences which have the problem of communication cost being very high. DELPHI, introduced in Section 3.2.2 is the most famous multi-party computation based privacy-preserving machine learning study and it was considered the most prominent result. Compared to DELPHI, the proposed method requires 13-14 times less communication cost, and compared to that in [5], 4-5 times less processing time with maintaining accuracy.

Table 3.5: Comparison to previous work for one CIFAR-100 image inference

Resnet-32	DELPHI(All-ReLU)	DELPHI(Best)	ICML	Proposed
Communication cost	$>>$ 8GB	$>>$ 2GB	Very low	597MB
Running time	240s	100s	3,942s	887s
Accuracy	68%	66%	69.43%	69.6%

3.4.4 Classification Using Library LATTIGO

LATTIGO is a homomorphic encryption library based on the Go language [49]. It offers significant advantages over other existing libraries, including SEAL, which was used in the previous experiments, particularly in terms of speed and ease of parameter modification. However, when considering the overall running time, along with the advantage of lower communication cost in SEAL, it is challenging to claim a clear advantage over DELPHI, as indicated in Table 3.5. To address this concern, experiments are conducted using the LATTIGO library for Proposed 1), and this dissertation presents the results obtained with the mentioned parameters in LATTIGO.

I explore the use of the LATTIGO encryption library, which allows for halving the size of the existing parameters from $N = 2^{16}$ and $n = 2^{15}$ to $N = 2^{15}$ and $n = 2^{14}$. As a result, the overall communication overhead is also reduced by half. A comparison is made with the previous benchmark paper, DELPHI, which exhibits a large communication overhead but benefits from lower online communication. Contrarily, our proposed LATTIGO-based method achieves improved accuracy (All-ReLU) while reducing both communication overhead and inference time compared to DELPHI. Furthermore, as mentioned earlier, the conventional level usage requiring 18 levels for ReLU is reduced to 15 by partitioning it into 5, 5, 5, resulting in reduced consumption. Additionally, we propose a novel technique, Proposed 3, which raises the level in a manner similar to replacing bootstrapping during ReLU. A comprehensive comparison of each proposed approach is presented in the table below.

Table 3.6: Inference using LATTIGO library

ResNet-32	Proposed 1(LATTIGO)	Proposed 1(LATTIGO)	Proposed 3(ReLU)	Proposed 3(ReLU)
Communication cost	597MB	298MB	1GB	500MB
Running time	283s	156s	240s	140s
Parameters	$N = 2^{16}, n = 2^{15}$	$N = 2^{15}, n = 2^{14}$	$N = 2^{16}, n = 2^{15}$	$N = 2^{15}, n = 2^{14}$

The inference time for a single image is approximately 156 seconds, and for 50 images, it requires 275 seconds, resulting in an amortized runtime of around 7 seconds. Furthermore, when compared to the baseline ResNet-32 accuracy, the achieved accuracy for 1000 images achieves 92.8%. These results demonstrate similar performance to previous findings while achieving approximately three times faster computation. At the current stage, further performance improvements can be expected by adjusting the special modulus and polynomial modulus and utilizing the communication environment to enhance the performance of comparison operations, rather than solely focusing on bootstrapping replacement.

Table 3.7: Classification runtime for one image using LATTIGO library

Image	Model	SEAL	LATTIGO
CIFAR-10	Resnet-20	526s	92s
	Resnet-32	850s	156s
	Resnet-44	1,167s	214s
	Resnet-56	1,480s	280s
	Resnet-110	2,898s	556s

Another noteworthy aspect lies in the amortized runtime measurement of the inference methods. In the case of SEAL, the best performance was achieved when 50 images were inferred simultaneously. However, in LATTIGO, it was observed that even with 100 images, further performance improvements were evident. This phenomenon is speculated to be attributed to the drastic reduction of the utilized levels, from 18 to

4, as observed in Proposed 3. The exact reasons behind this enhancement are currently under investigation and remain subject to further exploration.

This dissertation sheds light on the intriguing performance differences between SEAL and LATTIGO-based inference methods concerning amortized runtime. Specifically, the reduction in utilized levels in LATTIGO seems to play a significant role in achieving improved performance, though comprehensive insights into the underlying mechanisms require further research.

Table 3.8: Classification runtime for multiple images using LATTIGO library

Image	Model	Runtime	Armortized runtime
CIFAR-10(50 images)	Resnet-20	144s	2.88s
	Resnet-32	275s	5.50s
	Resnet-44	377s	7.54s
	Resnet-56	482s	9.64s
	Resnet-110	799s	15.98s
CIFAR-10(100 images)	Resnet-20	286s	2.86s
	Resnet-32	482s	4.82s
	Resnet-44	658s	6.58s
	Resnet-56	871s	8.71s
	Resnet-110	1,555s	15.55s

One notable distinction of LATTIGO is its refresh process starting from level 0, whereas SEAL leaves one level remaining. Apart from the slightly lower communication overhead advantage in SEAL, this difference is not significant.

In summary, the Lattigo library enables the utilization of smaller parameters compared to existing ones. However, achieving this requires designing a new multiplexed parallel convolution that differs from the approach used in previous works. Currently, efforts are underway to implement new code tailored to the current situation, aiming to maximize performance improvements and achieve superior outcomes compared to existing results.

3.5 Future Works

While the improvements and advantages have been discussed, there are still areas for further improvement and research. Firstly, previous studies have focused on larger parameters such as $N = 2^{16}$ and $n = 2^{15}$ due to bootstrapping requirements. However, my research can be conducted with smaller parameters. This has the potential to significantly reduce communication costs by approximately half, as demonstrated in the experimental results. Additionally, it is expected to reduce the overall experiment time. However, using smaller parameters may result in a reduction in depth and the ability to use fewer special modulus, potentially increasing the computation time for operations like multiplication.

Furthermore, the experiments in this study are conducted solely using a CPU, without utilizing GPU acceleration. By incorporating GPU acceleration, significantly faster inference times can be achieved compared to the current setup. This has the potential to yield superior results in all aspects compared to DELPHI.

One limitation of this study is that the replacement of the bootstrapping process in the ciphertext refresh prevents the use of the imaginary-removing bootstrapping method described in previous papers [5]. This poses a challenge when implementing deep networks like ResNet-110, as it may lead to the occurrence of the divergence phenomenon, which is an existing problem. However, this issue is planned to be addressed in future work, allowing for the implementation of imaginary removing even without the bootstrapping part.

In summary, the Lattigo library's advantage of facilitating easy modification of special modulus contributes to its faster speed compared to other libraries. Adjusting the polynomial modulus enables changes in the packing structure, resulting in benefits not only in computation time but also in communication costs during image inference.

It is expected to achieve approximately four times improvement in computation time and around two times improvement in communication costs. Furthermore, the library provides the potential to utilize communication-based bootstrapping as an alternative to comparison operations, offering additional opportunities for performance enhancement.

Chapter 4

Attack Algorithm for a Keystore-Based Secret Key Generation Method

4.1 Introduction

With the increasing prevalence of data storage and transmission in public cloud systems, ensuring the security of these systems has become of paramount importance. Traditional password-based approaches can leave users vulnerable to dictionary attacks, emphasizing the need for more secure alternatives. Secure keys have gained widespread adoption in various domains such as file encryption, virtual private network access, and user authentication, as they disclose less user information compared to password-based methods. However, conventional key generation methods face challenges when it comes to managing long-term files in cloud systems.

The requirement to individually encrypt each file with random secure keys presents a necessity for long-term data protection, considering the characteristics of long-term file storage and frequent user access. This approach helps maintain the security of cloud systems against potential attacks such as ciphertext-only attacks or chosen-plaintext attacks.

To address the goal of achieving secure encryption with a one-key-for-one-file paradigm, a novel key generation method utilizing the keystore seed was proposed. The proponents of this method claimed that it could generate a large number of information-theoretically secure keys. However, in this dissertation, I present a groundbreaking approach to breaking their key generation scheme by reconstructing the keystore seed using a small set of collected keys.

This chapter focuses on discussing the issues associated with the newly proposed method for solving the key generation problem in file transfer and storage within cloud systems. The previously proposed method could generate multiple keys using a minimal amount of information through the utilization of the keystore seed. The proponents argued that the generated keys were information-theoretically secure. However, I delve into the problems arising from the presence of duplicated keystore seeds and highlight the vulnerability of the method to attacks.

Subsequently, in Section 4.2, I outline the problems associated with generating and managing secret keys using the exclusive or operation. Section 4.3 delves into the linear attack algorithm employed for the secret key generation method presented in [46]. I explore how this attack leverages key index information and provide a numerical analysis, demonstrating the number of collected keys required to successfully compromise the proposed method. Moreover, I present the attack probability of the linear attack using tables and graphical figures, illustrating the potential weaknesses in the scheme.

Finally, in Section 4.4, I unveil the information-theoretic weaknesses of modified schemes that employ hashed keys, shedding light on the limitations and vulnerabilities of these alterations.

4.2 Problems of Generating and Management Using Exclusive-OR Keys

In previous papers, exclusive OR key generation using keystore seed, and if the management method is used, since the attacker does not know keystore seed, they claim that each user can obtain an information-theoretically secure key just by sending the index. In order words, even if the attacker knows the key, cannot find out information about other keys.

To explain about problems of key generation, see how it is generated when $t = 5$. After selecting one encryption key k_i, when the corresponding index is transmitted along with the ciphertext, the key index m_1, m_2, m_3, m_4, m_5 value can be known. And form this index, the lengths of l are summed to become k_i, and the five combined results also be seen. If $4l$ starting from m_1, m_2, m_3, m_4 are known, the value of l starting from m_5 is automatically known because the added value is known. This means that each key contains as much information as l bits of the keystore seed. If an attacker finds out information about multiple keys in the same way, could find out all of the keystore seed if the attacker exceeds a certain number of keys. This is because the entropy of k_j with respect to index j is not greater than $l(1 - \epsilon)$, even if the values and index values are known for n keys in the formula which is claimed in the previous paper, but is determined as one value for a given index entropy will be 0.

In special cases for one key, the number of possible candidates could be reduced. Among the index values for when $t = 5$, set as $X = m_5 - m_1$, satisfy $3 < X < l$ at this time there are 2^X possible candidates for keystore seed for all corresponding indexes. This value is much smaller than 2^{4l}, which is a real possible candidate in the general case, which can be seen as another problem due to overlapping use. If the attacker checks the case of $3 < X < l$, when generating one key, the frequency used for each value is shown in Table 4.1 below.

In this case, even if there is only one key it is possible to obtain a candidate group for the keystore seed with a small amount of operation. It has the advantage of being able to attack at an earlier time by checking the index value in multiple keys and checking whether it is satisfied in this special case first, and then checking other values based on the result.

Table 4.1: Frequency used for one key in special case

5 times use	$l - m_5 + m_1$
4 times use	$(m_5 - m_4) + (m_2 - m_1)$
3 times use	$(m_4 - m_3) + (m_3 - m_2)$
2 times use	$(m_3 - m_2) + (m_4 - m_3)$
1 time use	$(m_2 - m_1) + (m_5 - m_4)$

4.3 Linear Attack on Key Generation and Management

4.3.1 Linear Attack Algorithm

In this section, I propose an attack algorithm to reconstruct a keystore seed from a number of collected keys. For example, assume that I have some keys with $t = 5$ as presented in [46]. Each key has 5 indices and consists of 5 binary exclusive OR subkeys of length l starting at given indices. Each key can make $l \times L$ submatrix M_i shown on the left side of Figure 4.1. Each M_i consists of l indicator vectors to generate key k_i. For example, I have one key with index $i = (1, 3, 4, 6, 7)$. Then, the indicator vector e_1^0 is $0101101100 \cdots 00$ (All 0 except indices 1,3,4,6,7). Next, the indicator vector e_1^1 is a circular shift to the right of e_1^0. Rows of M_i consist of $e_i^0, \cdots, e_i^{l-1}$ and rank$(M_i) = l$ because it has l independent indicator vectors. If the $tl << L$ condition is not satisfied, there are dependent indicator vectors due to overlap by cyclic shift. The indicator matrix M is made by stacking up $M_i's$. Consequently, I stack up submatrices until M satisfies rank$(M) = L$. Finally, I find keystore seed using the system of linear equations as Figure 4.1 because M becomes full rank and it is invertible. The attack algorithm is summarized in Algorithm 4.1. If the indicator matrix M has rank L by stacking up several indicator submatrices, Argument 1 is not correct for a sufficiently large n to make M full rank. Thus, their key generation method is not secure.

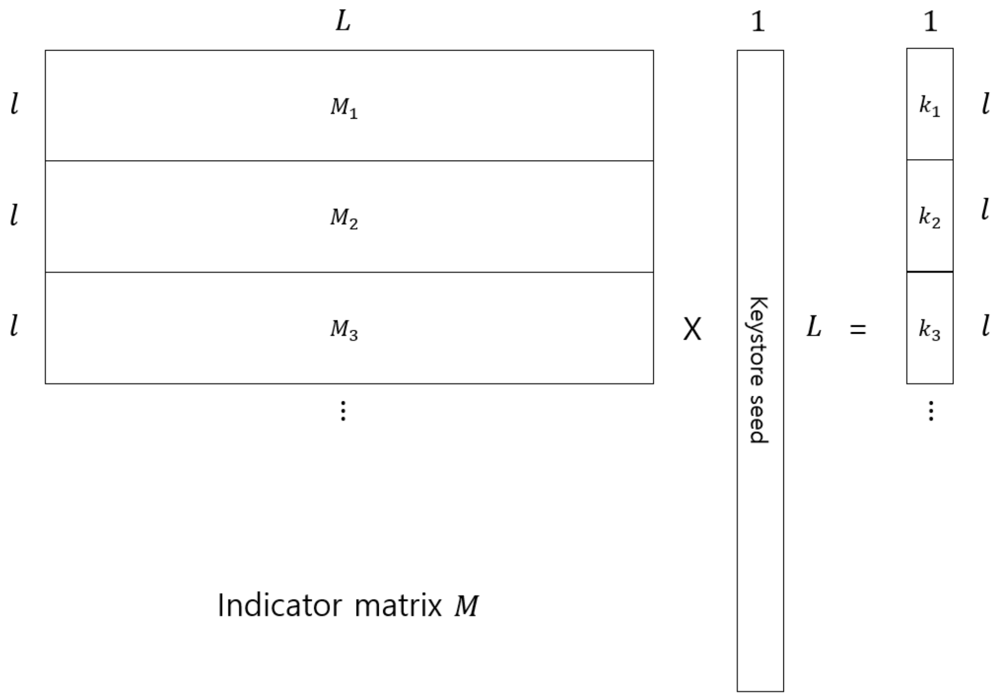


Figure 4.1: Matrix operation to find keystore seed.

Algorithm 4.1 Successful attack probability with R keys

```
Input: Variables $L, l, R, t$
Output: True if the indicator matrix rank is larger than or equal to $L$
  for $i$ from 1 to $R$ do
    $key\_index\_set \leftarrow$ Randomly select $t$ integers in range of (0, $L-1$)
    ${e_i}^0 \leftarrow$ indicator vector of $key\_index\_set$ of length $l$
    for $j$ from 1 to $l-1$ do
      ${e_i}^j \leftarrow$ circular cyclic shift right once of ${e_i}^{j-1}$
    end for
    $M_i \leftarrow stack\{{e_i}^0, \cdots, {e_i}^{l-1}\}$
    $M = stack\{M_1, \cdots, M_i\}$
    if rank$(M) \geq L$ then
      return True
    end if
  end for
```

Let Z be a random variable defined as

$$Z = \begin{cases} 1 & \text{if } rank(M) = L \\ 0 & \text{if } rank(M) \neq L. \end{cases}$$

With this random variable, the left-hand side of (2.1) can be rewritten as

$$\begin{aligned} &H(k_i \mid j_1, \cdots, j_n, i, k_{j_1}, \cdots, k_{j_n}) = \\ &\quad H(k_i \mid j_1, \cdots, j_n, i, k_{j_1}, \cdots, k_{j_n}, Z = 1)P(Z = 1) \\ &\quad + H(k_i \mid j_1, \cdots, j_n, i, k_{j_1}, \cdots, k_{j_n}, Z = 0)P(Z = 0), \end{aligned} \tag{4.1}$$

where $P(Z = 1)$ means that the keystore seed is reconstructed and the key's entropy goes to 0 because k_i is automatically determined with key index i. Therefore, (4.1) only contains the $P(Z = 0)$ case. Since $H(k_i \mid j_1, \cdots, j_n, i, k_{j_1}, \cdots, k_{j_n}) \leq l$, I have

$$\begin{aligned} &H(k_i \mid j_1, \cdots, j_n, i, k_{j_1}, \cdots, k_{j_n}, Z = 0)P(Z = 0) \\ &\qquad \leq lP(Z = 0). \end{aligned}$$

According to numerical analysis, $P(Z = 0)$ becomes almost 0 when the number of collected keys increases, which means that the lower bound of entropy in the n-th order expansion in Argument 1 is not correct for a large n. Although Argument 1 is correct for very small n, it is not useful in that they could not generate many secure keys because the purpose of their proposed method is to deal with one-key-for-one-file in cloud systems. In other words, when the entropy of the generated keys becomes 0, the keystore seed cannot be used to generate secure keys anymore. Thus, attackers can reconstruct the keystore seed with high probability, which means that their key generation method is no longer information-theoretically ϵ-secure. In the next subsection, I will show the number of collected keys to make rank$(M) = L$ by numerical analysis.

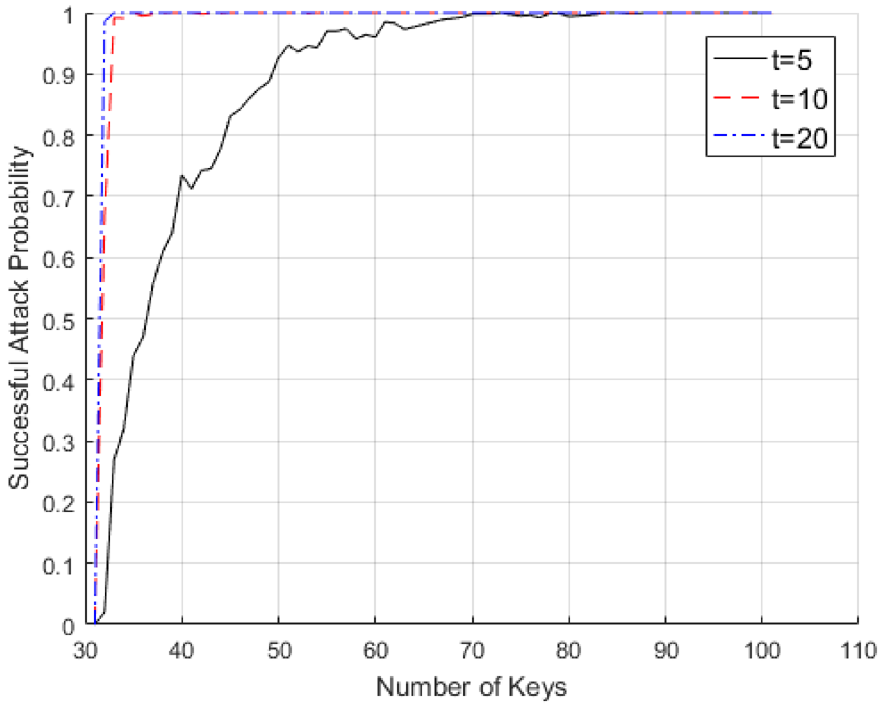
4.3.2 Successful Linear Attack Probability

The successful attack probability with R keys is given as a probability that an indicator matrix M has a rank larger than or equal to L by using R keys as in Algorithm 4.1. Clearly, at least L/l keys are required to make M with full rank. Figures 4.2 and 4.3 show that the successful attack probability of the key generation algorithm in [46] is numerically derived for $L = 2^{12}, 2^{14}, l = 2^7, 2^8$, respectively when $t = 5, 10, 20$. Table 4.2 lists the successful attack probability in Figures 4.2 and 4.3 for several numbers of R.

Table 4.2: Successful attack probability of the proposed attack algorithm

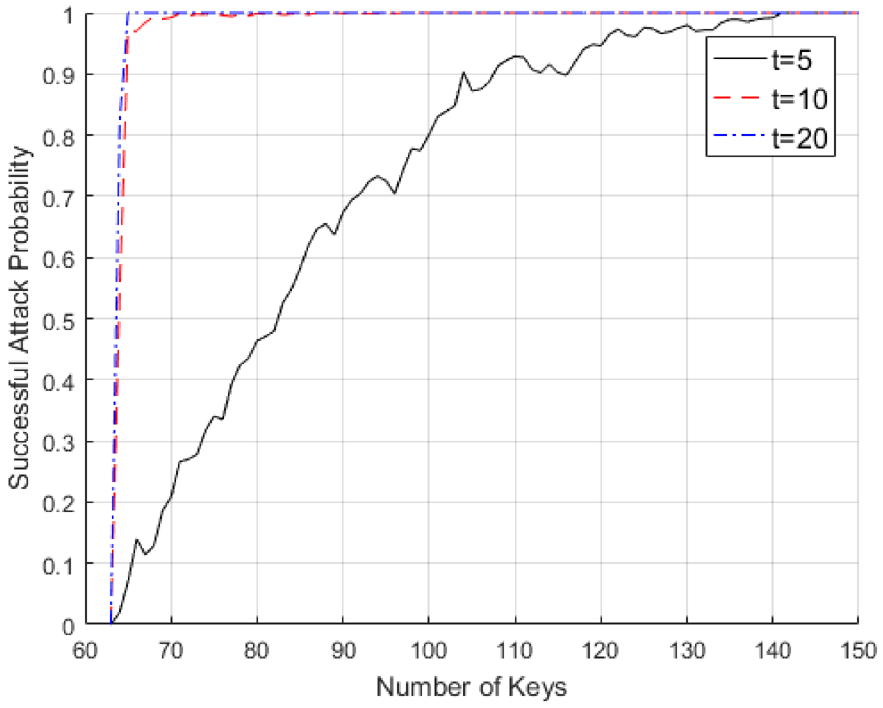
$L = 2^{12}, l = 2^7$			
Number of keys	$t = 5$	$t = 10$	$t = 20$
$R = 32$	0.02	0.664	0.986
$R = 40$	0.735	1	1
$R = 84$	1	1	1

$L = 2^{14}, l = 2^8$			
Number of keys	$t = 5$	$t = 10$	$t = 20$
$R = 64$	0.02	0.540	0.820
$R = 70$	0.208	0.992	1
$R = 100$	0.801	1	1
$R = 141$	1	1	1



(a)

Figure 4.2: Successful attack probability of the proposed attack algorithm when: $L = 2^{12}, l = 2^7$.



(b)

Figure 4.3: Successful attack probability of the proposed attack algorithm when: $L = 2^{14}, l = 2^8$.

4.4 Information Theoretic Weakness of Modified Yang-Wu's Schemes with Hashed Keys

The forward secrecy is a property such that if a secret key is compromised, past keys are not compromised. According to the key generation method in [46], several keys are generated from one keystore seed through a linear combination. If the number of generated keys is large enough, the newly generated key will have only a very small entropy from previously generated keys. This idea can be checked via the following observation.

For binary independent random variables X and Y, suppose that $H(X) = H(Y) = 1$ and $H(X,Y) = 2$. Then, I have

$$\begin{aligned} H(X,Y|X \oplus Y) &= H(X,Y,X \oplus Y) - H(X \oplus Y) \\ &= H(X,Y) - H(X \oplus Y) = 1. \end{aligned}$$

This can easily be extended and applied to Yang and Wu's algorithm intended to provide independent and uncorrelated secret keys for the one-key-for-one-file long-term secure system. Assume that I have one key generated from tl bits of keystore seed as in Figure 4.4. If I know the subkeys $K(m_j)K(m_j+1)\cdots K(m_j+l-1)$ for $j = 1,\cdots,t-1$, I can derive the subkey $K(m_t)K(m_t+1)\cdots K(m_t+l-1)$ since I know the key $k(0)k(1)\cdots k(l-1)$. As t increases, the number of subkeys generating a key becomes large. This becomes a weak point when giving the indicator matrix M a full rank in Section 4.3. As the simulation results show that the successful attack probability of the proposed attack algorithm for $t = 10, 20$ increases abruptly compared to $t = 5$ when the number of collected keys becomes large. In addition, the successful attack probability becomes very large as t increases. Therefore, a large value of t for the key generation scheme should be avoided.

In real applications, it is very important to provide a way of strong protection for the keystore seed. However, in a cloud environment, there is a possibility that some information can be disclosed during the processing such as key generation, file encryption, or decryption, due to undiscovered weaknesses of systems or side-channel attacks as in [47]. In this dissertation, I show that it is possible to reconstruct the entire keystore seed even if a very small number of generated keys (i.e., 84 keys) are leaked compared to the total size of the possible keys (i.e., 2^{53} keys).

In order to reduce the risk of keystore seed reconstruction, the encryption using a hashed key $h(k)$ was proposed in [48], where k is a generated key from the keystore seed and $h(\cdot)$ is a one-way hash function. It is true that encryption with a hashed key could avoid the proposed linear attack of keystore seed reconstruction. However, avoiding the linear attack does not guarantee information-theoretically ϵ-secure since hashed keys are the same number of bits as original keys. If the original keystore is not information-theoretically ϵ-secure, hashed keys are not also information-theoretically ϵ-secure since hashing is a one-to-one mapping. Hashing only increases computational complexity, but it does not guarantee key entropy.

The hashed key can be a countermeasure for the proposed linear attack. Moreover, by introducing a hash chain for key generation, it is possible to increase both the computational complexity of the linear analysis and the number of possible keys. Let us set each subkey as $a_j = K(m_j)K(m_j+1)\cdots K(m_j+l-1)$ for $j = 1, \cdots, 5$. Then, the key k_j is generated as $k_j = h(h(h(h(h(a_1) \oplus a_2) \oplus a_3) \oplus a_4) \oplus a_5)$, where a_i is a subkey and $h(\cdot)$ is a one-way hash function from $\{0,1\}^*$ to $\{0,1\}^l$. Note that if the order of applying a_i is changed, the generated key is completely different when a cryptographic hash function such as SHA-2 or SHA-3 is used. Even though this type of countermeasure cannot guarantee information-theoretically ϵ-secure keys, but it can be a cryptographically secure way.

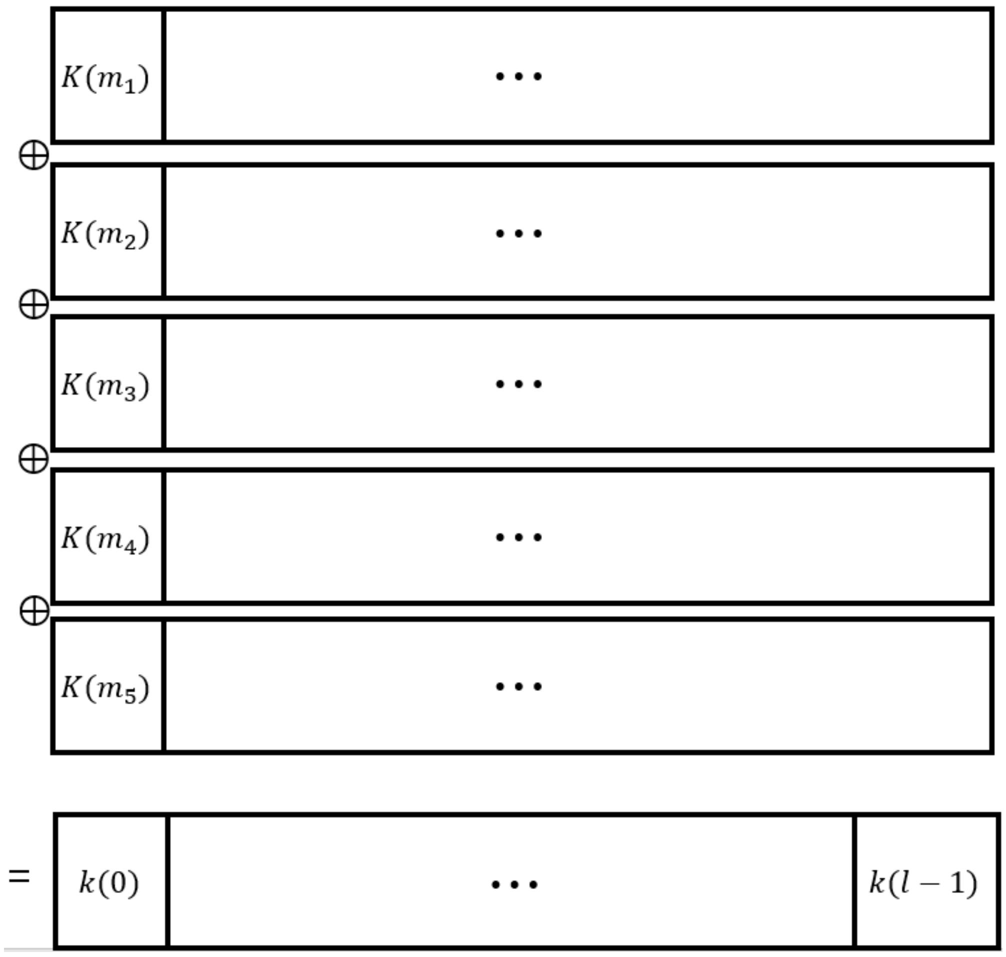


Figure 4.4: Key generation by subkeys.

Chapter 5

Conclusions

In this dissertation, research on privacy-preserving machine learning using a hybrid method of HE and MPC to reduce bootstrapping computation cost and attack algorithm for a keystore-based key generation method were presented.

In Chapter 2, a brief introduction of privacy-preserving machine learning and secret key generation method using keystore seed were briefly overviewed. Preliminaries for homomorphic encryption especially CKKS, multi-party computation, privacy-preserving machine learning, and secret key generation and management were presented.

In Chapter 3, I proposed a new method of bootstrapping replacement using communication cost. A bottleneck of HE-PPML could be solved using ciphertext refresh when the server and client could do communications. Random addition and deletion could execute with communication between the server and clients. Using the proposed technique, the bootstrapping replacement has good time and communication performance compared to the previous work. Since it is possible to improve performance using GPUs and parameter changes, it can be expected to outperform DELPHI in all aspects of performance.

In Chapter 4, I proposed the attack algorithm for a keystore-based secret key generation and management. As the demand for long-term data over the public clouds increases, a large number of secure keys are needed. To deal with this problem, Yang and Wu proposed a new key generation method using the keystore seed [46]. In this dissertation, I proposed an attack algorithm for their key generation method, where a small number of collected keys can be used to reconstruct the keystore seed with high probability. Although the encryption using a hashed key could avoid the proposed reconstruction attack, it still does not guarantee the information-theoretically ϵ-secure in certain situations where some information is leaked. Therefore, a new secure key generation method with keystore seed can be studied in future research.

Bibliography

[1] Yao, Andrew C. ”Protocols for secure computations,” 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982). IEEE, 1982

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

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

[4] Lee, E., Lee, J.-W., Kim, Y.-S., and No, J.-S. ”Optimization of homomorphic comparison algorithm on RNS-CKKS scheme,” IEEE Access, 10:26163–26176, 2022.

[5] Lee, Eunsang, et al. ”Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions,” International Conference on Machine Learning. ICML, 2022.

[6] Krizhevsky, Alex, and Geoffrey Hinton. ”Learning multiple layers of features from tiny images,” (2009): 7.

[7] Mishra, Pratyush, et al. ”Delphi: A cryptographic inference service for neural networks,” 29th USENIX Security Symposium (USENIX Security 20). 2020.

[8] Juvekar, Chiraag, Vinod Vaikuntanathan, and Anantha Chandrakasan. ”GAZELLE: A low latency framework for secure neural network inference,” 27th USENIX Security Symposium (USENIX Security 18). 2018.

[9] Cheon, Jung Hee, et al. ”A full RNS variant of approximate homomorphic encryption,” Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25. Springer International Publishing, 2019.

[10] https://github.com/Microsoft/SEAL, November 2020. Microsoft Research, Redmond, WA.

[11] Gentry, Craig. ”Fully homomorphic encryption using ideal lattices,” Proceedings of the forty-first annual ACM Symposium on Theory of Computing. 2009.

[12] Gentry, Craig, Shai Halevi, and Nigel P. Smart. ”Better bootstrapping in fully homomorphic encryption,” International Workshop on Public Key Cryptography. Springer, Berlin, Heidelberg, 2012.

[13] Cheon, Jung Hee, et al. ”Homomorphic encryption for arithmetic of approximate numbers,” International Conference on the Theory and Application of Cryptology and Information Security. Springer, Cham, 2017.

[14] Damgård, Ivan, et al. ”Multiparty computation fromsomewhat homomorphic encryption,” Annual Cryptology Conference(CRYPTO). Springer, Berlin, Heidelberg, 2012

[15] Chillotti, Ilaria, et al. ”TFHE: fast fully homomorphic encryption over the torus,” Journal of Cryptology 33.1 (2020): 34-91.

[16] Mohassel, Payman, and Yupeng Zhang. ”Secureml: A system for scalable privacy-preserving machine learning,” 2017 IEEE symposium on security and privacy (SP). IEEE, 2017.

[17] Gilad-Bachrach, Ran, et al. ”Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy,” International Conference on Machine Learning. ICML, 2016.

[18] J. Liu, M. Juuti, Y. Lu, and N. Asokan. “Oblivious Neural Network Predictions via MiniONN Transformations,” In: CCS ’17

[19] Mohassel, Payman, and Peter Rindal. ”ABY3: A mixed protocol framework for machine learning,” Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018.

[20] Rouhani, Bita Darvish, M. Sadegh Riazi, and Farinaz Koushanfar. ”Deepsecure: Scalable provably-secure deep learning,” Proceedings of the 55th Annual Design Automation Conference. 2018.

[21] Boemer, Fabian, et al. ”nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data,” Proceedings of the 16th ACM International Conference on Computing Frontiers. 2019.

[22] Boemer, Fabian, et al. ”nGraph-HE2: A high-throughput framework for neural network inference on encrypted data,” Proceedings of the 7th ACM Workshop on Encrypted Computing Applied Homomorphic Cryptography. 2019.

[23] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song, ”Bootstrapping for approximate homomorphic encryption,” EUROCRYPT 2018

[24] Bossuat, J.-P., Mouchet, C., Troncoso-Pastoriza, J., and Hubaux, J.-P. ”Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys,” In EUROCRYPT 2021, pp. 587–617. Springer, 2021

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

[26] Cheon, J. H., Han, K., Kim, A., Kim, M., and Song, Y. ”Bootstrapping for approximate homomorphic encryption,” In EUROCRYPT 2018, pp. 360–384. Springer, 2018b.

[27] Jung, W., Kim, S., Ahn, J. H., Cheon, J. H., and Lee, Y. ”Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPU,”. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(1):114–148, 2021

[28] Lee, J.-W., Lee, E., Lee, Y., Kim, Y.-S., and No, J.-S. ”Highprecision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function,” In EUROCRYPT 2021, pp. 618–647. Springer, 2021c.

[29] Lee, J.-W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.-S., et al. ”Privacypreserving machine learning with fully homomorphic encryption for deep neural network,” IEEE Access, 10:30039–30054, 2022b

[30] Lee, Y., Lee, J.-W., Kim, Y.-S., Kang, H., and No, J.-S. ”High-precision approximate homomorphic encryption by error variance minimization,” In EUROCRYPT 2022, pp.551–580. Springer, 2022c

[31] L´opez-Alt, A., Tromer, E., Vaikuntanathan, V.: ”On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption,” In: Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, pp. 1219–1234. ACM (2012)

[32] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: ”(Leveled) fully homomorphic encryption without bootstrapping,” In: Proceedings of ITCS, pp. 309–325. ACM (2012)

[33] Brakerski, Z., Vaikuntanathan. V.: ”Efficient fully homomorphic encryption from (standard) LWE,” In: Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, pp. 97–106. IEEE Computer Society (2011)

[34] Brakerski, Z., Vaikuntanathan, V.: ”Fully homomorphic encryption from ring-LWE and security for key dependent messages,” In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011).

[35] Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: ”Fully homomorphic encryption over the integers,” In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010).

[36] Fan, J., Vercauteren, F.: ”Somewhat practical fully homomorphic encryption,” IACR Cryptology ePrint Archive 2012/144 (2012)

[37] Naehrig, M., Lauter, K., Vaikuntanathan, V.: ”Can homomorphic encryption be practical?,” In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, pp. 113–124. ACM (2011)

[38] Gentry, C., Sahai, A., Waters, B.: ”Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based,” In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013).

[39] Q. Lou and L. Jiang, ‘‘SHE: A fast and accurate deep neural network for encrypted data,’’ in Proc. Adv. Neural Inf. Process. Syst. (NIPS), 2019, pp. 1–9

[40] J. Fan and F. Vercauteren, ‘‘Somewhat practical fully homomorphic encryption,’’ Cryptol. ePrint Arch., Bellevue, WA, USA, Tech. Rep. 2012/144, 2020. [Online]. Available: https://eprint.iacr.org/ 2012/144

[41] C. Shannon, “Communication theory of secrecy systems,” Bell System Technical Journal 28 (4): 656 – 715, 1949.

[42] U. Maurer, ”Conditionally-Perfect Secrecy and a Provably-Secure Randomised Cipher,” Journal of Cryptology, vol. 5, no.1, pp. 53-66, 1992

[43] Morris, R.; Thompson, K. Password security: A case history. *Commun. ACM* **1979**, *22*, 594–597.

[44] Monrose, F.; Reiter, M.K.; Li, Q.; Wetzel, S. ”Cryptographic key generation from voice,” In Proceedings of the 2001 IEEE Symposium on Security and Privacy, 2001; pp. 202–213.

[45] Menezes, A.J.; van Oorschot, P.; Vanstone, S. *Handbook of Applied Cryptography*. CRC Press: Boca Raton, FL, USA, 1996.

[46] Yang, E.H.; Wu, X.W. ”Information-theoretically secure key generation and management,” In Proceedings of 2017 IEEE International Symposium on Information Theory (ISIT), Aachen, Germany, 25–30 June 2017; pp. 1529–1533.

[47] Bazm, M.-M.; Lacoste, M.; Sudholt, M.; Menaud, J.-M. ”Side Channels in the Cloud: Isolation Challenges, Attacks, and Countermeasures,” 2017. Available online: https://hal.inria.fr/hal-01591808/ (accessed on 17 February 2019)

[48] Wu, X.W.; Yang, E.H.; Wang, J.H. ”Lightweight security protocols for the Internet of Things,” In Proceedings of the 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, Canada, 8–13 October 2017.

[49] Mouchet, Christian Vincent, et al. ”Lattigo: A multiparty homomorphic encryption library in go,” Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography. No. CONF. 2020.

[50] Al Badawi, Ahmad, et al. ”OpenFHE: Open-source fully homomorphic encryption library,” Proceedings of the 10th Workshop on Encrypted Computing Applied Homomorphic Cryptography. 2022.

초록

이 학위 논문에서는 다음 두 가지의 연구가 이루어졌다: i) 정보보호 머신러닝에서 통신 환경을 사용하여 bootstrapping을 대체한 암호문 새로고침 방법 ii) 키 스토어 시드 기반 비밀 키 생성의 문제점 및 공격 방식.

첫 번째로, 동형암호를 이용한 정보보호 머신러닝의 가장 큰 문제점인 bootstrapping 시간이 매우 크다는 단점을 해결하기 위해 통신 환경을 이용하였다. 랜덤한 값을 더해서 유저에게 보내준 후 유저가 복호화 및 재 암호화를 통해 암호문의 레벨을 올려주어 bootstrapping 효과를 대체할 수 있고, 기존 연구들에 비해 시간 및 통신량 측면에서도 좋은 성능을 유지하면서 pre-trained 네트워크에서도 사용가능한 것을 확인하였다. 또한 중요한 파라미터인 정확도 측면에서도 기존 결과와 유사한 값을 유지하면서 다른 문제점 없이 성능 개선을 하였음을 확인하였다.

두 번째로, 키 스토어 시드 기반 비밀 키 생성 방법의 문제점과 그에 대한 선형 공격을 제기한다. 확률적으로 작은 개수의 키가 모였을 때에도 공격이 가능한 것을 확인하여 그에 맞는 수식적인 문제도 제기하였다. 마지막으로 그 선형 공격을 막을 수 있는 방식을 제공하며 기존의 문제점에 대한 해결책도 제시하였다.

주요어: 완전동형암호, 다자간계산, 정보보호 머신러닝, 부트스트래핑, 암호학, 암호문 새로고침, 정보이론적으로 안전한 키, 키 생성, 키 저장방식, 키스토어 시드, 데이터 프라이버시

학번: 2017-22828