



Degree Project in Communication Systems

Second cycle, 30 credits

# Confidential Federated Learning with Homomorphic Encryption

Master's Thesis Degree Project

**ZEKUN WANG**



# **Confidential Federated Learning with Homomorphic Encryption**

**Master's Thesis Degree Project**

ZEKUN WANG

Master's Programme, Communication Systems, 120 credits  
Date: November 21, 2023

Supervisors: Nicolae Paladi, Roberto Guanciale

Examiner: Amir H. Payberah

School of Electrical Engineering and Computer Science

Host company: CanaryBit AB

Swedish title: Konfidentiellt federat lärande med homomorf kryptering

Swedish subtitle: Examensarbete för masterexamen



## Abstract

Federated Learning (FL), one variant of Machine Learning (ML) technology, has emerged as a prevalent method for multiple parties to collaboratively train ML models in a distributed manner with the help of a central server normally supplied by a Cloud Service Provider (CSP). Nevertheless, many existing vulnerabilities pose a threat to the advantages of FL and cause potential risks to data security and privacy, such as data leakage, misuse of the central server, or the threat of eavesdroppers illicitly seeking sensitive information. Promisingly advanced cryptography technologies such as Homomorphic Encryption (HE) and Confidential Computing (CC) can be utilized to enhance the security and privacy of FL. However, the development of a framework that seamlessly combines these technologies together to provide confidential FL while retaining efficiency remains an ongoing challenge. In this degree project, we develop a lightweight and user-friendly FL framework called **Heflp**, which integrates HE and CC to ensure data confidentiality and integrity throughout the entire FL lifecycle. Heflp supports four HE schemes to fit diverse user requirements, comprising three pre-existing schemes and one optimized scheme that we design, named Flashev2, which achieves the highest time and spatial efficiency across most scenarios. The time and memory overheads of all four HE schemes are also evaluated and a comparison between the pros and cons of each other is summarized. To validate the effectiveness, Heflp is tested on the MNIST dataset and the Threat Intelligence dataset provided by CanaryBit, and the results demonstrate that it successfully preserves data privacy without compromising model accuracy.

## Keywords

Cloud Technology, Confidential Computing, Federated Learning, Homomorphic Encryption, Trusted Execution Environment



## Sammanfattning

Federated Learning (FL), en variant av Maskininlärning (ML)-teknologi, har framträtt som en dominerande metod för flera parter att samarbeta om att distribuerat träna ML-modeller med hjälp av en central server som vanligtvis tillhandahålls av en molntjänstleverantör (CSP). Trots detta utgör många befintliga sårbarheter ett hot mot FL:s fördelar och medför potentiella risker för datasäkerhet och integritet, såsom läckage av data, missbruk av den centrala servern eller risken för avlyssnare som olagligt söker känslig information. Lovande avancerade kryptoteknologier som Homomorf Kryptering (HE) och Konfidentiell Beräkning (CC) kan användas för att förbättra säkerheten och integriteten för FL. Utvecklingen av en ramverk som sömlöst kombinerar dessa teknologier för att erbjuda konfidentiellt FL med bibehållen effektivitet är dock fortfarande en pågående utmaning. I detta examensarbete utvecklar vi en lättviktig och användarvänlig FL-ramverk som kallas Heflp, som integrerar HE och CC för att säkerställa datakonfidentialitet och integritet under hela FL-livscykeln. Heflp stöder fyra HE-scheman för att passa olika användarbehov, bestående av tre befintliga scheman och ett optimerat schema som vi designar, kallat Flashev2, som uppnår högsta tids- och rumeffektivitet i de flesta scenarier. Tids- och minneskostnaderna för alla fyra HE-scheman utvärderas också, och en jämförelse mellan fördelar och nackdelar sammanfattas. För att validera effektiviteten testas Heflp på MNIST-datasetet och Threat Intelligence-datasetet som tillhandahålls av CanaryBit, och resultaten visar att det framgångsrikt bevarar datasekretessen utan att äventyra modellens noggrannhet.

## Nyckelord

Molnteknik, Konfidentiell databehandling, Federerad inlärning, Homomorfisk kryptering, Betrodd körningsmiljö



## Acknowledgments

I would like to express my sincere gratitude to the following individuals and groups for their support and help throughout the completion of my degree project.

I would like to thank my company supervisor Nicolae Paladi from CanaryBit for your guidance, expertise, and continuous support during this academic journey. You provided me with this valuable chance to work on such an interesting topic and guided me on both academic writing and technical development. Your guidance and insights have been invaluable in improving the quality of my work. I also want to thank Prof. Paul Stankovski Wagner from Lund University, Nicolae's colleague, for giving me helpful advice and inspiration in the biweekly meetings. I really appreciate the CanaryBit team for supplying the VM instances, datasets, and other resources that are critical for me to conduct my development and experiments.

I would also like to thank Roberto Guancia from KTH for your valuable insights, feedback, and encouragement, which powered me up and broadened my vision. I extend my appreciation to Prof. Amir H. Payberah from KTH for taking the time to evaluate and provide feedback on this degree project.

As this thesis referred to a lot of existing works and research, I would also like to acknowledge the academic community and developers, without whom this research would not have been possible. Thank you for inspiring me through your work and dedication.

Finally, to my dear friends and family, your encouragement and patience have been a constant source of my motivation. I could not have completed this degree project without your love and understanding.

Stockholm, November 2023

Zekun WANG



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.2	Research Question . . . . .	3
1.3	Objectives . . . . .	4
1.4	Research Methodology . . . . .	4
1.4.1	Information Collection . . . . .	4
1.4.2	Framework Developing . . . . .	5
1.4.3	Framework Testing . . . . .	5
1.4.4	Advanced Strategy Exploration . . . . .	6
1.4.5	Report Writing . . . . .	6
1.5	Delimitations . . . . .	6
1.6	Evaluation & News Value . . . . .	6
<b>2</b>	<b>Background</b>	<b>9</b>
2.1	Confidential Computing . . . . .	9
2.1.1	AMD Secure Encrypted Visualization . . . . .	10
2.1.2	Security Challenges of AMD SEV . . . . .	11
2.2	Homomorphic Encryption . . . . .	11
2.2.1	Homomorphic Encryption Algorithms . . . . .	12
2.2.2	Limitations of Homomorphic Encryption . . . . .	13
2.3	Federated Learning . . . . .	14
2.3.1	Federated Learning Algorithms . . . . .	15
2.3.2	Secure Aggregation . . . . .	17
2.3.3	Comparison of FL frameworks . . . . .	18
2.4	SSL/TLS . . . . .	20
2.5	Threat Intelligence . . . . .	21
2.5.1	Threat Intelligence Sharing . . . . .	22
2.6	Related Work . . . . .	23

<b>3</b>	<b>Methodology</b>	<b>29</b>
3.1	Threat Model . . . . .	29
3.2	Framework Developing Methods . . . . .	32
3.2.1	Startup . . . . .	32
3.2.2	Implementing HE . . . . .	33
3.2.3	Preliminary Tests . . . . .	33
3.3	Framework Testing . . . . .	33
3.3.1	Planned Test Settings . . . . .	34
3.3.2	Evaluation Metrics . . . . .	34
3.3.3	Datasets . . . . .	35
3.3.3.1	TI dataset . . . . .	35
3.3.3.2	MNIST dataset . . . . .	35
3.4	Prototype Deployment . . . . .	36
<b>4</b>	<b>Framework Design</b>	<b>37</b>
4.1	System Overview . . . . .	37
4.2	Federated Learning Process . . . . .	39
4.2.1	Stage 1: Initial Setup . . . . .	39
4.2.2	Stage 2: FL Training . . . . .	39
4.2.3	Stage 3: Completion . . . . .	41
4.3	Homomorphic Encryption Schemes . . . . .	41
4.3.1	Flashe . . . . .	41
4.3.2	Flashev2 . . . . .	43
4.3.3	CKKS & BFV . . . . .	46
4.4	Quantization and Weighted Averaging . . . . .	46
4.4.1	Symmetric Quantization . . . . .	46
4.4.2	Primary Weighted Averaging Strategy . . . . .	48
4.4.3	Improved Weighted Averaging Strategy: MWAvg . . . . .	49
4.4.4	Flashev2+MWAvg . . . . .	50
<b>5</b>	<b>Implementation</b>	<b>53</b>
5.1	HeFlp Submodules . . . . .	53
5.1.1	heflp.secureproto . . . . .	53
5.1.1.1	homoencrypschemes . . . . .	54
5.1.1.2	quantization . . . . .	55
5.1.2	heflp.training . . . . .	55
5.1.3	heflp.client . . . . .	55
5.1.4	heflp.strategy . . . . .	56
5.1.5	heflp.utils . . . . .	56

5.1.6	heflp.info . . . . .	57
5.2	HeFlp Usage . . . . .	57
5.3	Deployment . . . . .	58
<b>6</b>	<b>Results and Evaluation</b>	<b>61</b>
6.1	Security Assessment . . . . .	61
6.2	Performance Evaluation . . . . .	63
6.2.1	Error Rates of HE Schemes . . . . .	63
6.2.2	Training Performance Evaluation . . . . .	65
6.2.2.1	Simulation settings . . . . .	65
6.2.2.2	Training Curves and Analysis . . . . .	65
6.3	Efficiency Evaluation . . . . .	69
6.3.1	Evaluation Settings and Metrics . . . . .	69
6.3.2	Time Efficiency Test of Uniform pattern . . . . .	70
6.3.3	Time Efficiency Test of Nonuniform Pattern . . . . .	72
6.3.4	Time Efficiency Test of Model Types . . . . .	74
6.3.5	Memory Overhead Test . . . . .	74
6.3.6	HE Efficiency Comparison . . . . .	77
<b>7</b>	<b>Conclusions and Future Work</b>	<b>79</b>
7.1	Discussion . . . . .	79
7.2	Future Work . . . . .	80
	<b>References</b>	<b>81</b>



# List of Figures

1.1	The General goal of this project is to enable multiple parties to collaborate on model training in a cloud context while retaining data privacy. Additional security mechanisms are required to protect data integrity and avoid leaking sensitive information. . . . .	3
2.1	FL architecture . . . . .	14
2.2	An example of threat intelligence sharing. . . . .	23
3.1	Two numerical solutions . . . . .	30
4.1	System overview . . . . .	38
4.2	Federated Training process with Homomorphic Encryption, red marks mean that these steps are optional and only necessary for some of the HE schemes . . . . .	40
4.3	An example of Flashe. Use Flashe to calculate the sum of two pictures securely. . . . .	42
4.4	An example of Flashev2. Use Flashev2 to calculate the sum of two pictures securely. . . . .	44
4.5	Two numerical solutions . . . . .	48
4.6	Combine MWAvg and HE encryption together. . . . .	51
5.1	Usage of Heflp . . . . .	58
6.1	collusion of the server and client adversaries. . . . .	62
6.2	Two numerical solutions . . . . .	66
6.3	Two numerical solutions . . . . .	67

6.4	Results of the time efficiency test when all the clients have the same weight 1000. The time costs of encryption, decryption, and aggregation (calculation) are recorded while the total time cost is the sum of three, representing the total time cost of one client. . . . .	71
6.5	Results of the time efficiency test when the weights of clients are in range [1000, 5000]. . . . .	73
6.6	Results of the time efficiency test for three models: CNN (50890 parameters), LSTM (220355 parameters), ResNet18 (11689512 parameters). . . . .	75
6.7	Results of the memory overhead test for three models: CNN (50890 parameters), LSTM (220355 parameters), ResNet18 (11689512 parameters). <i>None</i> means the original size of the model without using any HE scheme . . . . .	76

# List of Tables

2.1	Comparison between different FL frameworks . . . . .	18
4.1	Supported homomorphic encryption schemes . . . . .	41
5.1	Metadata transmitted from clients to server . . . . .	56
5.2	Configurations transmitted from server to clients . . . . .	57
6.1	The settings of the uniform test . . . . .	64
6.2	The settings of the nonuniform test . . . . .	64
6.3	Settings for the simulation of FL on MNIST or TI dataset. . . . .	68



## List of acronyms and abbreviations

AMD-SP	AMD Secure Processor
BFV	Brakerski-Fan-Vercauteren
BGV	Brakerski-Gentry-Vaikuntanathan
CA	Certificate Authority
CC	Confidential Computing
CKKS	Cheon-Kim-Kim-Song
CNN	Convolutional Neural Network
CSP	Cloud Service Provider
DHC	Distributed Homomorphic Cryptosystem
DP	Differential Privacy
FATE	Federated AI Technology Enabler
FHE	Fully Homomorphic Encryption
FL	Federated Learning
FRL	Federated Reinforcement Learning
FTL	Federated Transfer Learning
FV	Fan-Vercauteren
HE	Homomorphic Encryption
Heflp	Homomorphic Encryption Federated Learning FLower Plugin
HFL	Horizontal Federated Learning
IBM FL	IBM Federated Learning
IDB	Signed Identity Block
IOC	indicator of compromise
LSTM	Long Short-Term Memory
MAC	Message Authentication Code
MHE	Multiparty Homomorphic Encryption
ML	Machine Learning
MPC	Multi-Party Computation
MSE	Mean Squared Error

OpenFL	Open Federated Learning
PHE	Partially Homomorphic Encryption
PPFL	Privacy-Preserving Federated Learning
PRF	Pseudorandom Function
REE	Rich Execution Environment
RMP	Reverse Map Table
SA	Secure Aggregation
SEV	Secure Encrypted Visualization
SEV-ES	SEV Encrypted State
SEV-SNP	SEV Secure Nested Paging
SGD	Stochastic Gradient Descent
SME	Secure Memory Encryption
SS	Secure Sharing
SSL	Secure Sockets Layer
SWHE	Somewhat Homomorphic Encryption
TEE	Trusted Execution Environment
TFF	TensorFlow-Federated
TI	Threat Intelligence
TLS	Transport Layer Security
VCEK	Versioned Chip Endorsement Key
VFL	Vertical Federated Learning
VM	Virtual Machine
ZKP	Zero-Knowledge Proofs

# Chapter 1

## Introduction

### 1.1 Background

Federated Learning (FL) [1] is one decentralized machine learning variant that aims to handle cases where training data is sensitive and distributed on different devices or parties. One main concern of FL is to train the global model collaboratively on the decentralized training data while protecting data privacy. In FL, multiple parties train their model locally based on their datasets and then aggregate all these partial models to obtain one global model which is expected to gain higher accuracy. In this way, the parties avoid transferring their sensitive data when collaborating.

For the typical FL architecture, there is one central server used to assist the FL process for aggregation. All the parties send their updated local models to the central server for aggregation. The central server, which could be deployed to one Cloud Service Provider (CSP), performs as a bridge between multiple parties. However, one deceptive CSP might abuse the data uploaded by the participants without being detected, and the participants could only choose to trust the CSP. Another problem is that it is hard for participants to verify that the aggregation is processed as expected on the central server side. For instance, one malicious central server may abuse the received models to extract sensitive data. It could cheat participants by stating that the aggregation is done but actually, it does not. So the behavior of CSP is invisible to a certain extent, making it hard to guarantee data security and privacy. In this scenario, there is a lack of bidirectional trust between the CSP and participants [2].

Trusted Execution Environment (TEE) [3] is one hardware-based technology that provides one isolated, verifiable, and user-controlled environment for Confidential Computing (CC). By using a built-in cryptographic system

and additional mechanisms, CSPs are prevented from tampering with or exposing the data or application protected by TEE. TEE helps to reset the trust relationship between end-users and CSPs by enabling end-users to control their remote execution environments reliably and securely. However, TEE also suffers from some security issues such as kernel attacks, side-channel attacks, or architectural attacks [4], which can violate the security guarantees of TEE especially data confidentiality. What's more, the TEE system itself is fully controlled by the CSP, thus if it is under some kind of attack and not secure anymore, it is difficult for users to realize such risks in time.

Homomorphic Encryption (HE) [5] is one cryptographic primitive that allows one or more operations, such as addition or multiplication, on encrypted data without decryption, protecting sensitive data from exposure during the whole process of computation. HE can guarantee the confidentiality of data. However, it is not designed to provide data integrity. Without integrity protection, the message might be tampered with during transmission or even in the storage of the server. On the other hand, HE does not provide the integrity of code or algorithm as well. Without additional mechanisms, it is impossible to ensure that aggregation runs as expected on the server side.

By leveraging Federated Learning and Homomorphic Encryption with TEE-based Confidential Computing together, this paves one possible way to scalable and secure multi-party data collaboration. CC and HE are complementary. In the scenario of FL, TEE is able to enhance the data and code integrity that HE lacks, while HE can provide a stronger guarantee of data confidentiality considering the various security vulnerabilities that TEEs have. By leveraging TEE and HE together with FL, it is promising to achieve stronger security and privacy towards customer data and workloads through the entire lifecycle.

However, existing gaps in the protocol stack and tooling slow down the wider adoption of CC and HE in cloud settings. Most of the existing FL frameworks don't support HE or only support 1 or 2 schemes with few concerns about their efficiency (see more details in Chapter 2). And there is a lack of study about deploying FL with HE in TEEs. Different techniques have different pros and cons, requiring a well-designed framework to leverage all these primitives properly. Moreover, The use of additional security mechanisms results in reduced efficiency and performance due to increased computational requirements and encryption noise. A higher level of security and guarantee of privacy always suffers from higher resource consumption and extra overhead [6]. Therefore, it is necessary to trade-off between efficiency and security to fit specific demands.

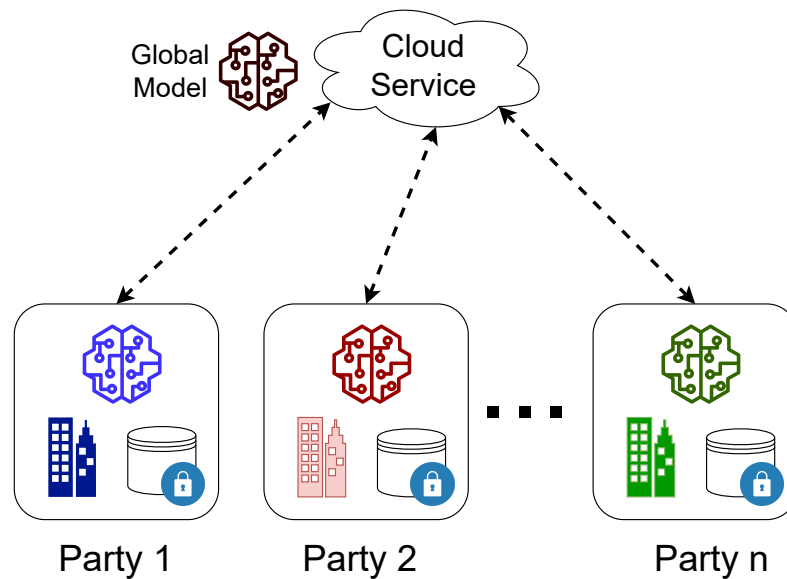


Figure 1.1: The General goal of this project is to enable multiple parties to collaborate on model training in a cloud context while retaining data privacy. Additional security mechanisms are required to protect data integrity and avoid leaking sensitive information.

## 1.2 Research Question

As mentioned in Section 1.1, several technologies protect data privacy during the data collaboration between multiple parties.

FL gives one decentralized approach for Machine Learning (ML) on distributed training data. HE allows computations on encrypted data, thus enhancing the data confidentiality in the whole process. TEE can be used to achieve CC in the CSP side. FL is the fundamental approach for data collaboration while HE and TEE are two technologies to enhance data security and privacy. But all of them have their limitations, as introduced in Section 1.1. It is hard to guarantee the security and privacy of data collaboration by any of them alone.

In Section 3.1, we define the threat model this project considers, including the adversaries and their capabilities. Therefore, the research question of this thesis project is: **Does combining the confidentiality guarantees of HE with integrity guarantees of TEEs protect the security of data in the FL setting from the predefined adversaries?**

The developed confidential FL framework is expected to provide strong data security and privacy during the whole lifecycle, which means that the

data confidentiality and integrity should be enhanced during both the data transmission between the server and participants and the data processing on the CSP side. On the other hand, it is supposed to retain the model performance with acceptable extra computation and communication overhead caused by the security mechanisms.

## 1.3 Objectives

The purpose of this project is to enhance data security and privacy in the FL context by applying HE and TEE. To achieve it and answer the research question, we expect to make the following objectives one by one.

1. Conduct one background pre-study of FL, HE, and CC, including one survey of existing Federated Learning frameworks, focusing on support of HE and other security mechanisms as well.
2. Design one FL framework combined with TEE and find HE scheme(s) that support required operations with acceptable overhead.
3. Implement the prototype with hardware-based TEE and test it in different FL scenarios (i.e. various models and datasets).
4. Formulate and implement a set of relevant benchmarks to evaluate the performance of the proposed framework.

## 1.4 Research Methodology

To achieve the objectives, this section defines several detailed tasks, each of which includes several steps and challenges. The corresponding methods that we plan to apply are also described here.

### 1.4.1 Information Collection

To explore the existing FL frameworks and figure out if they support Homomorphic Encryption well, this thesis will look through not only web resources (such as the tutorials and technical documents) but also other related surveys and papers. The survey will also include relevant extensions and plugins. As there are a lot of frameworks around FL, the open-source and prevalent (more stars and forks on Github\*) frameworks will be considered

---

\*Github: <https://github.com/>

a higher priority, such as FATE [7], Flower [8], OpenFL [9], TensorFlow Federated [10]. In this way, we are able to figure out the shortcomings of existing frameworks and possible improvements. It also helps to collect more information which may provide inspiration in the future as well. The literature review is also conducted in this step.

### 1.4.2 Framework Developing

To give our own solution, this project will use one surveyed framework as a start, and our solution will be added as one extension to it. Some frameworks such as Federated AI Technology Enabler (FATE), TensorFlow-Federated (TFF) already support some kinds of HE schemes, but there has been no combination with confidential computing. And Plugins such as BatchCrypt [11] are out of maintenance. To decide the FL framework that the project will be based on, several aspects are considered:

1. If it is easy to start developing and convenient to deploy.
2. If it has existing HE support or available HE extensions.
3. If it is convenient and flexible to add new features, i.e. by extensions or plugins.
4. If it has well-organized documents and tutorials.

More details about the developing methods are in Section 3.2.

### 1.4.3 Framework Testing

After developing the framework prototype, it will then be applied and tested in a scenario where multiple organizations or enterprises share the information for better collaboration. The tests will consider different FL scenarios, such as different models, client numbers, or datasets. Metrics include model accuracy, computational speed, and memory occupation compared to the plaintext. Some further improvements may be applied based on the evaluation results.

The resulting prototype will be deployed in a real TEE. In our project, the implementation will rely on x86 platforms accessed using the AWS platform equipped with AWS SEV\*.

For more details of metrics and test settings, please refer to Section 3.3.

---

\*AWS SEV-SNP: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html>

#### **1.4.4 Advanced Strategy Exploration**

To optimize the system performance and security, this project will conduct an evaluation of various HE schemes and compare their pros and cons. It is also promising to optimize existing HE schemes to make them fit the needs of FL training better.

In this step, we hope to develop strategies that select proper HE schemes, and FL algorithms considering the demands of security and efficiency.

#### **1.4.5 Report Writing**

As the project proceeds, we keep recording and writing down the findings, and finally, we will arrange and reorganize these findings as one report. The report needs to not only include all the details but also analyze and give conclusions.

### **1.5 Delimitations**

Because of the limitation of time and resources, this project mainly focuses on the application instead of the algorithms themselves. The project will not develop totally new HE algorithms or FL algorithms. Besides, in this project we do not plan to dive deeply into the principles of TEE or compare the performance of different TEE products. Instead, we focus on the application of it and deploy our aggregation service on it. For more details about the usage of TEE, its advantages and disadvantages and so on, we suggest to refer the survey we did in Section 2.1, where some relevant works provides more comprehensive information about it.

This project will only focus on the scenarios that CanaryBit is able to provide, which means that the framework development mainly considers some specific scenarios. Although we strive for flexibility, testing and improvement across a wide range is challenging.

### **1.6 Evaluation & News Value**

The main contribution of this project will be the framework that we are going to develop. After developing the framework, we are also going to test and evaluate its performance by multiple metrics in different FL scenarios. For more details, please refer to Section 3.3.

The baseline is to successfully combine HE and TEE with FL and implement the prototype to the given scenarios. The new framework should

enhance the security level and give an evaluation of its efficiency. After reaching the baseline, it's possible to improve the framework by applying new strategies or algorithms. For example, strategies for choosing proper HE scheme or FL aggregation algorithm for a flexible trade-off between efficiency and security performance.

The project will utilize HE to enhance the security and data privacy of the FL system in a confidential computing context. The prototype will be implemented in a real confidential computing environment (TEE), which will prove the feasibility of our framework. Nowadays people are putting more attention on the data security and user privacy preservation in the context of ML and cloud technology. Secure collaboration of ML is becoming more essential with the increase of applications relying on ML and FL. Policies are formulated requiring companies to have certain protection of sensitive information as well. All these factors lead to an urgent demand of additional secure mechanisms for distributed model training. Therefore, people who are looking for more secure FL solutions may be interested in this work.

Besides, the project also surveys other popular frameworks, which could be helpful when someone needs to choose the framework for certain security concerns.



# Chapter 2

## Background

### 2.1 Confidential Computing

CC is one new technology that aims to process data in a confidential and secure manner. It's proposed to protect the sensitive data in use from unauthorized access or exposure. According to the definition given by the Confidential Computing Consortium, CC is the technology that provides "the protection of data in use by performing computation in a hardware-based, attested Trusted Execution Environment" [12]. TEE is one hardware-based cryptographic technique that creates one secure and isolated execution environment where the code and data within it are protected. TEE is commonly defined to provide at least three security properties: data integrity, data confidentiality, and code integrity. The code running in one TEE can not be tampered with and the data in use in the TEE is protected from being leaked or tampered with, thus fulfilling the security requirements of confidential computing well [13, 14]. Some technique details of TEE are introduced as follows.

The property of isolation assures confidentiality as the applications running in TEE are isolated from each other and the rest of the system, or called Rich Execution Environment (REE). The sensitive data is stored in one secure area in memory, thus the data confidentiality is also guaranteed [15].

The data and code integrity are protected via two properties of TEE. The execution environment including its runtime and memory is isolated from other applications, therefore preventing unauthorized modifications from outside. What is more, TEE includes one feature called remote attestation (remote verification) which allows TEE to prove its trustworthiness for remote users. More in detail, the TEE is able to generate one certificate that contains information about TEE and the data within it, such as the process id, the

application code, and the underlying hardware. This certificate is signed by the embedded private key inside the hardware and could be verified by the CA [16]. In this way, the user is able to make sure that the code is running as expected inside TEE, thus the integrity is achieved.

There are multiple TEE hardware architectures provided by different vendors, such as Intel SGX [17], AMD SEV [18], and ARM TrustZone [19].

### 2.1.1 AMD Secure Encrypted Visualization

This project implements AMD Secure Encrypted Visualization (SEV) for TEE protection. In this section technique details of SEV is discussed.

AMD SEV technologies include three generations of SEV, which is designed for Virtual Machine (VM) isolation. The first generation of SEV only implements Secure Memory Encryption (SME) to isolate VM instances in a memory level, which means that each VM is assigned a dedicated AES key for memory encryption. In this way, the data in use is automatically encrypted in memory and all the other components such as any other VM or the hypervisor are not able to read the plaintext data without a key. However, only SME is not enough even for data confidentiality since the CPU register state is still public. The second SEV generation SEV Encrypted State (SEV-ES) is then proposed to add a higher level of protection. In SEV-ES, the CPU register state of each VM is encrypted when it stops running so that the data in use inside the CPU is also under protection. SEV-ES provides a stronger guarantee for confidentiality and a certain level of integrity since it is also able to check if the CPU register state is changed by any malicious components. The third generation of SEV is called SEV Secure Nested Paging (SEV-SNP). In SEV-SNP, memory integrity protection is achieved, which is not provided by either SEV or SEV-ES. SEV-SNP is then able to protect VMs from malicious hypervisor-based attacks, such as data replay or memory re-mapping. SEV-SNP achieves memory integrity protection by adding a mechanism called Reverse Map Table (RMP), which tracks the ownership of each memory page. With RMP, each page of memory can only be accessed by one specific owner, and only the owner is able to write it. And of course, the RMP itself is not writeable by software. Besides, SEV-SNP also adds several optional functionalities to handle specific security issues such as side channel attacks [20, 21]. Since SEV-SNP provides both data confidentiality and integrity protections, SEV-SNP fulfills the requirements of CC. In this project, the third SEV generation SEV-SNP is used as the TEE to provide CC.

AMD SEV also provides remote attestation so that a third party could

verify the data comes from the specific VM. To get a valid attestation report, the VM firstly sends the attestation request with a block of data  $\mathcal{M}$  to the AMD Secure Processor (AMD-SP) via a protected path. Then the AMD-SP uses the Versioned Chip Endorsement Key (VCEK) to sign an attestation report which includes the data  $\mathcal{M}$ , system information and the Signed Identity Block (IDB) information. IDB is generated during the VM launch stage and contains the VM identifier and launch digest. However, SEV and SEV-ES only support attestation in the guest VM launch stage, while SEV-SNP allows flexible attestation. In SEV-SNP, an attestation report could be requested at any time from the AMD-SP by the VM[20]. In this project, the remote attestation could be used to guarantee the integrity of models and the security for exchanging keys when building the secure communication channel between participants and VM.

### 2.1.2 Security Challenges of AMD SEV

Although AMD SEV is designed to fulfill the requirements of CC, it is always facing security challenges and might be vulnerable to some attacks violating its security guarantees, especially confidentiality. For instance, AMD SEV allows outside access to read its encrypted VM memory, which might allow CPU-side attacks that violate the data confidentiality by monitoring its memory updates [21]. Li et al [22] recently found another vulnerability of AMD SEV that exploits ciphertext side channels to steal private keys, allowing adversaries to infer private execution estates or recover the plaintext.

Although AMD SEV is always facing new security issues, AMD is updating its design to address these issues constantly. In this project, we assume AMD SEV is at least secure enough to protect the data and code integrity. By running the FL aggregation service inside the TEE, the participants are able to verify that the aggregation is working as expected. Addressing the security issues of TEE is out of the scope of this project.

## 2.2 Homomorphic Encryption

Homomorphic Encryption (HE) is one cryptographic technology that can do computations on encrypted data directly [5], which is in contrast to the classic encryption, where the data has to be firstly decrypted before performing any computation. HE could be defined by the following equation:

$$E(m_1) \oplus E(m_2) = E(m_1 \oplus m_2), \forall m_1, m_2 \in M \quad (2.1)$$

where  $E$  is the encryption method,  $\oplus$  is the supported operation and  $M$  is the message domain.

The process of performing computation on the encrypted data is called homomorphic evaluation. The confidentiality of the data is preserved during the homomorphic evaluation and the result is in an encrypted form. The final result of the expected operations could be obtained by decrypting the result of homomorphic evaluation.

## 2.2.1 Homomorphic Encryption Algorithms

There are several variants of HE, but they all satisfy the definition 2.1. According to the level of supporting evaluation operations, HE could be classified as three main types: Fully Homomorphic Encryption (FHE), Somewhat Homomorphic Encryption (SWHE), and Partially Homomorphic Encryption (PHE) [23, 24, 25].

FHE is the most powerful type of HE since it supports arbitrary computations and an unlimited number of evaluation operations on the encrypted data. However, it also has the highest computation and memory overhead. FHE algorithm example is Gentry's bootstrapping scheme [26], which uses ideal lattices as plaintext to support arbitrary homomorphic operations.

SWHE allows some types of operations a finite number of times. SWHE is not as powerful as FHE, but more efficient and practical in real use cases since it does not cost as much computation and memory resource as FHE. Examples are Fan-Vercauteren (FV) scheme [27] and Cheon-Kim-Kim-Song (CKKS) scheme [28].

FV algorithm supports both addition and multiplication in the integer plaintext space while CKKS algorithm allows complex numbers as plaintext and supports both addition and multiplication, too. However, CKKS only supports approximate computation with a given precision. It is also necessary to note that by applying the bootstrapping technique, CKKS could become one FHE scheme [29, 30].

PHE allows only one type of operation but has no limit to the number of times, with the highest efficiency and lowest overhead compared to FHE or SWHE. An example is Paillier [31], which only supports addition operation and takes integers as plaintext.

Considering that these algorithms have different pros and cons, the trade-offs need to be done based on the requirements and preferences of specific use cases. FHE is suitable for the scenario where functionality is more considered

than efficiency, while PHE is the best choice when only one type of operation is required [23]. For this project, different HE schemes may be used for different settings. And it also depends on which FL aggregation algorithm is used. For example, if the aggregation requires computation on float numbers, then CKKS could be suitable, or if the classic FedAvg algorithm is used, one PHE scheme that only supports addition operation is enough. The choice of HE scheme follows the principle of least functionality to reduce costs and maximize efficiency, which means the scheme that fulfills the requirements and does not have redundant features is preferred [6].

## 2.2.2 Limitations of Homomorphic Encryption

Homomorphic Encryption (HE) is powerful and promising for a wide range of applications. However, it also suffers from several limitations, making it challenging to implement. In this section, the limitations that this project concerns are mainly discussed. Briefly, the efficiency and security concerns are mostly considered in this project.

Considering the extra cost caused by HE, the processing speed of HE evaluation is very slow and the cost of computational resources and memory could be many more times higher than direct computation on plaintext data. For instance, CKKS algorithm could be more than 10000x slower than equivalent operations on plaintext [6]. In a FL setting, the encrypted data is going to be sent to the server, thus the communication cost also matters. As the ciphertext is always larger than plaintext, e.g. FHE the ciphertext size could be over 1000x larger than plaintext [6], HE also causes extra communication overhead. In general, the more operations the HE algorithm supports, the higher overhead it suffers [32, 6, 33].

Another limitation is that HE is only able to protect the confidentiality of data, but can not provide integrity by itself, which means that it can not guarantee that the encrypted data has not been tampered with by an attacker or adversary. However, this could be patched by some additional mechanisms, such as checksums [34]. And there's also a recent research conducted by A. Viand et al. [13] that shows the possible approaches to provide integrity by leveraging HE with some other cryptographic primitives together, such as Message Authentication Code (MAC), Zero-Knowledge Proofs (ZKP) or TEE attestation.

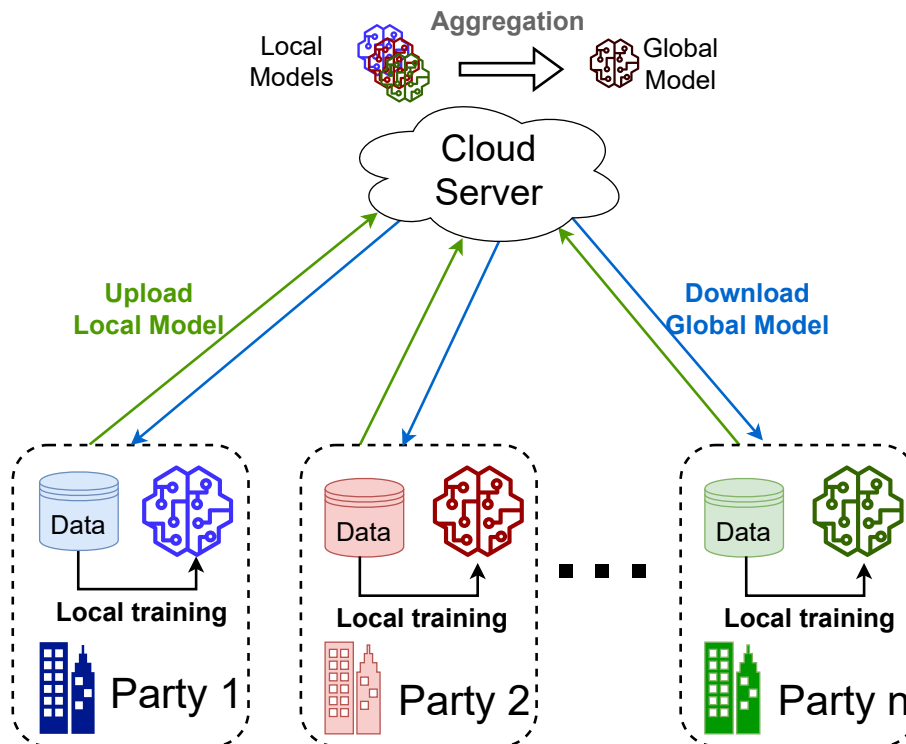


Figure 2.1: FL architecture

## 2.3 Federated Learning

Federated Learning (FL) is one variant of the machine learning approach that allows multiple clients to collaborate on training one more powerful global model without sharing their raw training data with each other or the central server, which means that the data remains on local devices and the global model is trained in a decentralized manner by only sharing intermediate results [1]. The classic architecture of FL is shown in Figure 2.1. FL has the ability to train global models on decentralized data and retain data privacy and security at the same time. Since the training data is kept locally, sensitive raw data, such as medical records or personal private information, could be protected from exposure. Another advantage of FL is that it can help reduce the communication overhead by completing parts of training locally and only transmitting model parameters or gradients instead of raw data. To be brief, FL especially benefits the scenarios where the distributed training data is sensitive or too large to be sent to the central server [35].

There are also several types of FL, according to how data is distributed and

how the collaboration is conducted [35, 36, 37]:

Horizontal Federated Learning (HFL) is the most general type for the case where the data samples in different parties have similar features but different indices [1, 38]. In HFL all the participants are likely to be the same type. For example, multiple hospitals collaborate to train one global model for COVID-19 image processing. In this case, the images from different hospitals have the same features but belong to different patients. And HFL is the main FL type considered in this project. Except HFL, several other FL variants are also important but are not used as widely as HFL. Vertical Federated Learning (VFL) takes the case that the data samples from different parties have different features but similar indices [39], i.e. from the same group of people. Federated Transfer Learning (FTL) is designed for one more complicated scenario where data samples of different parties have different features and different indices as well [40]. Federated Reinforcement Learning (FRL) handles the demand that different parties need to leverage Reinforcement Learning to learn from their own actions and environment rewards [41, 42].

Considering HFL, each iteration of the training process is divided into 3 steps [43]. First, all the participants fetch the current global model from the central server and prepare a local dataset to initialize local training. In this step, all the participants have the same initial model but different training datasets. And the machine learning algorithm for local training is decided. Second, local training is performed on each party's local dataset to refine the model using the decided algorithm. In this step, each participant trains the model to adjust to their own data. Third, all the participants send the refined models back to the central server for global aggregation. The server uses a specific FL algorithm to aggregate the parameters of all participants' models and update the global model. Then the updated model is sent back to each participant to trigger the next iteration until the global model converges.

FL is promising and able to be applied to a lot of applications in different areas, such as image recognition [44], healthcare [45, 36] or autonomous vehicles [35]. In this project, FL is the basic setting for data collaboration and all the techniques (HE and CC) mentioned above provide additional security protections.

### 2.3.1 Federated Learning Algorithms

Various Federated Learning algorithms are proposed for aggregating. In this section, the most widely used two algorithms are introduced, and these two will be mainly considered and leveraged in this project.

*FedAvg* is the classic FL algorithm proposed by B. McMahan et al. in 2017 [1]. The goal of *FedAvg* is to minimize the global objective function:

$$\min f(w) = \sum_{k=1}^K p_k F_k(w) \quad (2.2)$$

where  $w$  is the global model parameters,  $K$  is the number of participating devices,  $p_k$  is the weight value and  $F_k$  is the objective function of device  $k$ .  $p_k$  could be  $\frac{1}{K}$  for a non-weighted version or  $\frac{n_k}{\sum_k n_k}$  for a weighted version where  $n_k$  is the size of training data for device  $k$ .

To achieve this, *FedAvg* is designed as the following steps. Assume there are totally  $N$  participants, for each iteration  $t$ ,  $K$  of them are selected to collaborate in updating the global model. Then each of these  $K$  participants uses Stochastic Gradient Descent (SGD) algorithm [46] to optimize the downloaded global model  $w_t$ , and the objective function for participant  $k$  is  $F_k$ . After a certain number of epochs, the updates  $w_{t+1}^k$  are then sent back to the central server and averaged in a weighted manner to obtain the updated global model  $w_{t+1}$ .

*FedAvg* is already proven to be practical and widely used in real applications [35]. However, it is not as powerful as expected to handle the systems and statistical heterogeneity of data. Systems heterogeneity is caused by the characteristics of each participant (i.e. available computational resources and memory). Statistical heterogeneity means the non-identically distributed training data between participants. To handle this issue, *FedProx* is designed by T. Li et al. in 2020 [47].

*FedProx* follows the same procedure as *FedAvg* but makes some modifications to the optimizer and the partial objective function of each participant. Each  $f_k$  is added by one proximal term to balance the variable partial update like this:

$$f_k(w^k, w_t) = F_k(w^k) + \frac{\mu}{2} \|w^k - w_t\|^2 \quad (2.3)$$

Based on the new objective function, *FedProx* defines one  $\gamma_k^t$ -inexact minimizer for local optimization:  $w_{t+1}^k = \arg \min f_k(w^k, w_t)$ . A  $w^*$  is considered as one  $\gamma_k^t$ -inexact solution when  $\|\nabla f_k(w^*, w_t)\| \leq \gamma_k^t \|\nabla f_k(w_t, w_t)\|$ , where  $\nabla f_k(w, w_t) = \nabla F_k(w) + \mu(w - w_t)$ .

In this project, *FedAvg* is the first priority and if time allows, *FedProx* will also be realized.

### 2.3.2 Secure Aggregation

Although FL is designed for the protection of local sensitive data, there are still some security issues. Recent research shows that even the exposure of model parameters may cause a leakage of sensitive information which should be protected from being leaked. Attackers could perform membership inference attacks to create multiple "shadow models" that imitate the functionality of the victim model and use these "shadow models" to extract information from the training data used by the victim model [48]. To enhance data privacy and protect confidentiality, some improvements are performed on FL.

Privacy-Preserving Federated Learning (PPFL) is one type of FL focusing on data privacy and security of participants. Besides collaborating on training the model by aggregating local models of all participants, Privacy-Preserving Federated Learning (PPFL) adds additional mechanisms to enhance a higher level of data privacy and security. In PPFL, various advanced cryptographic technologies are leveraged to protect the sensitive data in different levels of security and in different stages [49, 50]. Secure Aggregation (SA) is one important component of PPFL, which allows model aggregation in a secure manner where the local updates are not exposed to any other participant or the central server during the global model aggregation [51, 52].

There are various techniques that could be used for Secure Aggregation (SA), and one common idea is to add well-designed noise to the local model parameters before they are sent to the central server. In this way, the detailed information of the specific model is hidden. One advanced cryptographic technique called Differential Privacy (DP) could be utilized for such noise generation. However, this method could suffer from a decrease in model accuracy because of the additional noise or mask [53]. Another idea is to combine Homomorphic Encryption (which is introduced in Section 2.2) with Federated Learning, which means using HE to encrypt the model parameters before sending it and letting the central server to perform the homomorphic evaluation operations for model aggregation [50]. In this way, the local models are aggregated without decryption, thus the data confidentiality is enhanced. Relying on Homomorphic Encryption, some more advanced techniques such as Multiparty Homomorphic Encryption (MHE) could be used to gain an even higher level of data privacy [54]. There is another cryptographic technique called Multi-Party Computation (MPC), which represents the protocols that are designed for multiple parties to collaborate on computing one function based on inputs from all the participants without exposing the private input to any other party. HE is one possible cryptographic primitive to achieve Multi-

Party Computation (MPC). However, MPC protocols could also be designed based on other cryptographic primitives such as Secure Sharing (SS). The difference is that HE is focusing on arbitrary functions while limiting the types of operations, however, MPC is more about computing the specific function and no need to support other operations [52]. Generally, all of these methods involve advanced cryptographic methods to enhance privacy and security.

To gain better data privacy, it is hard to avoid additional overheads and effects on model quality. On the one hand, the additional cryptographical mechanisms result in high computational overhead and memory occupation, slowing down the training process [6]. On the other hand, as mentioned before, it is hard to retain the same quality of model as well as a high level of data privacy due to the additional noise, especially when the scale of FL is large. How to solve or relieve these problems remains challenging [55]. In most cases, it is necessary to make trade-offs between performance and data security.

### 2.3.3 Comparison of FL frameworks

There are various Federated Learning frameworks available for use, and in order to choose one appropriate framework for this project, one survey is conducted to compare different FL frameworks focusing on their support of Homomorphic Encryption. In this project, a total of six open-source FL frameworks are surveyed and compared. The results are arranged and listed in Table 2.1.

Name	FL Algo Exp	Supported HE	HE Schemes	Extensions	Organization	Github Stars
FATE[7]	FedAvg HeteroLR HeteroNN	PHE, SWHE	Paillier CKKS FV	BatchCrypt FLASHE	WeBank	5k
TFF[10]	FedAvg FedSGD FedProx	PHE, SWHE, FHE	Paillier CKKS BFV/BGV	TF-Encrypted TF-SEAL	Tensorflow	2.1k
PySyft[56]	FedAvg	PHE, SWHE, FHE	Paillier CKKS BFV/BGV	TenSEAL	OpenMined	8.8k
Flower[8]	FedAvg FedProx FedBN	×	×	×	Flower	2.7k
IBM FL[57]	FedAvg FedProx FedAdam	Unknown	Unknown	×	IBM	423
OpenFL[58]	FedAvg FedProx FedOpt	×	×	×	Intel	546

Table 2.1: Comparison between different FL frameworks

Federated AI Technology Enabler (FATE) is an industrial grade FL framework developed by WeBank. It not only supports Federated Learning but has some additional functions for better data security. It has secure computation components that support Homomorphic Encryption and Multi-Party Computation, however, with some limitations. FATE framework embedded in one Paillier cryptosystem and also supports other secure techniques such as secret sharing. With the help of some available extensions, such as FLASHE [59] or BatchCrypt [11], FATE is able to enable more powerful HE schemes such as FV or CKKS.

TensorFlow-Federated (TFF) is another popular FL framework designed for facilitation of FL research and experimentation. Not like FATE, TFF enables developers to simulate the FL process easily and see the results. TFF itself does not support HE explicitly. But there are some well-developed assistant extensions. With the help of these extensions, TFF is able to leverage HE for secure aggregation. More specifically, TFF is able to achieve PHE with the help of TF-Encrypted extension [60], which supports Paillier HE scheme. By using the TF-SEAL extension [61], TFF is able to leverage Microsoft SEAL, which is a powerful FHE library that supports both FHE (Brakerski-Gentry-Vaikuntanathan (BGV), Brakerski-Fan-Vercauteren (BFV)) and SWHE (CKKS).

PySyft is one multi-language library that supports easy Federated Learning. It is an extension of Pytorch and enables some additional functions for data security. There's one library combined with PySyft called TenSEAL [62] that extends the Microsoft SEAL library to enable tensor operations and enhance the HE capability. With the help of TenSEAL, PySyft gains the functionality of HE and supports all the schemes available in the SEAL library such as CKKS, BFV, and BGV.

The three frameworks introduced above support HE by themselves or by some available extensions. Besides, another three frameworks that are not as popular as them are also surveyed.

Flower is a friendly Federated Learning framework with nice tutorials and flexible design. One advantage of Flower is that it is able to combine with any Machine Learning framework like Pytorch, TensorFlow, or JAX. Flower seems to not support HE. However, considering it's very flexible and convenient for configuration, it's possible to combine existing HE libraries, such as TenSEAL [62] and Pyfhel [63] into it. Although it does not support HE by itself, indeed there is one extension called Salvia [64] that enables it to support secure aggregation by SecAgg protocol, which is one MPC protocol [65].

IBM Federated Learning (IBM FL) is a Python framework for FL in an enterprise environment. IBM Federated Learning (IBM FL) actually support HE according to their documents. It's also mentioned that the user is able to set four levels of encryption, which provide different levels of security and precision. Higher-level encryption achieves better security and precision but requires higher resource consumption. However, the specific HE scheme is not mentioned in their documents, thus more technique details remain unknown.

Open Federated Learning (OpenFL) is also one Python 3 framework for FL. However, OpenFL neither supports HE nor has available extensions of HE.

In this project, Flower is finally chosen to set up the FL environment, and the TEE and HE are implemented based on it. Flower is very flexible and easy to learn, compared to other frameworks like FATE or TEE. It is the newest framework, having a rapid increase of Github stars, which shows its potential. It is open-source, and has comprehensive documents, especially code examples for different settings, making it easy and convenient to learn and start developing quickly. Another important point is that Flower is lightweight and focuses on FL, while FATE and PySyft have many redundant functions for this project. Although it does not support HE by itself, it is very convenient to combine existing HE libraries into it since it provides one Numpy client that allows developers to add operations to the data in standard Numpy format. On the contrary, the existing plugins of FATE are out-of-date and can not be implemented easily because of the lack of documents. Although TEE and PySyft have available HE extensions, because of their complexity, it is relatively harder to develop new extensions or modify existing ones.

There are actually other frameworks like Clara\* developed by Nvidia, but they are either not open-source or not widely used.

## 2.4 SSL/TLS

The Secure Sockets Layer (SSL) and its successor Transport Layer Security (TLS) are cryptographic protocols for secure communication over a network. They are designed to establish secure connections between parties and provide confidentiality, integrity, and authenticity in the data transmission. SSL [66] was firstly proposed in 1990s and quickly evolved into TLS [67] in 1999. In recent 20 years TLS has grown across several versions (the latest version is TLS 1.3) and became the mainly-used protocol for secure internet

---

\*Clara: <https://developer.nvidia.com/blog/federated-learning-clara/>

communication [68].

To establish an SSL/TLS connection between a client (send the request to initiate) and a server (wait and respond to a request from the client), a handshake protocol is designed to negotiate the TLS version to be used, encryption algorithms, exchange keys, and certificates and build a secure session [69]. For authentication, SSL/TLS allows the client and server (Optional) to verify the identity of each other by a certificate signed by a Certificate Authority (CA). The certificates of the server and client provide authenticity when setting up the connection and prove that the server and client are valid entities. In the process of handshaking, asymmetric encryption is used for exchanging symmetric keys (master secret) used for encrypted data transmission. Symmetric encryption ensures the confidentiality of data and achieves a higher efficiency compared to asymmetric encryption by using the same key for both encryption and decryption.

Data transmitted on the encrypted link is divided into records, each comprising a header and payload. To protect the data integrity, a Message Authentication Code (MAC) is generated using the shared symmetric key and attached to the message, detecting any invalid modification during the transmission [69].

SSL/TLS is widely used in modern networks and has been applied to various applications (protocols), such as HTTPS [70] for web browsing, IMAP/POP3 with TLS [71] for email services, and so on. In this project, SSL/TLS is utilized to enhance the communication security between clients and the aggregation server.

## 2.5 Threat Intelligence

To evaluate the performance of the proposed framework in this project and demonstrate its feasibility, the framework targets to train one ML model for Threat Intelligence (TI).

Threat Intelligence (TI) is one important component that helps to detect network intrusions and security vulnerabilities [72]. With the increasing sophistication of cyber attacks, traditional security measures, such as firewalls, antivirus software, or other signature-based detection systems, are no longer sufficient enough [72, 73]. In this context, TI becomes more and more important to provide cybersecurity defense, faster incident response, and aid in making decisions. TI refers to one process that collects and analyzes data from multiple sources to identify potential cyber threats and relieve the compromise. The objective of TI is to help network administrators of organizations make

informed decisions and realize the risk landscape. By leveraging the rich threat information and intelligent data processing methods such as Machine Learning (ML), TI is able to handle more complex security issues that traditional methods can not deal with, such as zero-day attacks [73].

The data that TI utilizes comes from a wide range of sources, such as the logs of computers, open-source datasets, or malware repositories. The collected data is analyzed to find the patterns or trends that could indicate cyber threats or potential vulnerabilities, such as malware infections, remote command execution, or advanced persistent threats (APTs). To analyze large volumes of data, TI utilize ML algorithms to extract patterns and correlations from the mass data automatically. ML models are able to detect the evidence that humans might miss and also retain the ability to recognize new security threats [73]. Through this data collection and analysis, a TI system can identify potential risks and take appropriate measures to handle them on time. One use case of TI is an anomaly detection system based on one ML model trained on Sysmon log data that records user activities on Windows computers. Sysmon is one Windows system service that logs detailed Windows system activities, such as process creations or network connections. After training the model on such a dataset, it is able to detect anomalous events and signal potential security incidents, which could alert on the dashboard [74].

Besides, with TI the administrators are able to process the anomalous events and obtain reasonable indicator of compromises (IOCs) easily, i.e. suspicious IP addresses or URLs, malicious file hashes, and source email addresses. We can feed firewalls, gateways, or other security event management systems with the obtained IOCs. Dashboard applications can visualize and index them as well.

### **2.5.1 Threat Intelligence Sharing**

As mentioned above, TI is data-based and requires analysis of comprehensive data to gain intelligence. Therefore, TI information sharing and collective learning as shown in Fig. 2.2 lead to many benefits. Considering that each organization has different data, some samples might be unique. The TI sharing allows establishing TI on the data from sources of all participating organizations. Therefore, it is able to cover more potential cyber threats with higher accuracy and speed up the identification compared to the local TI based on data only from certain organization [73].

Another advantage of TI sharing is to improve the collaboration between organizations to face new threats, reducing the likelihood of cascading effects

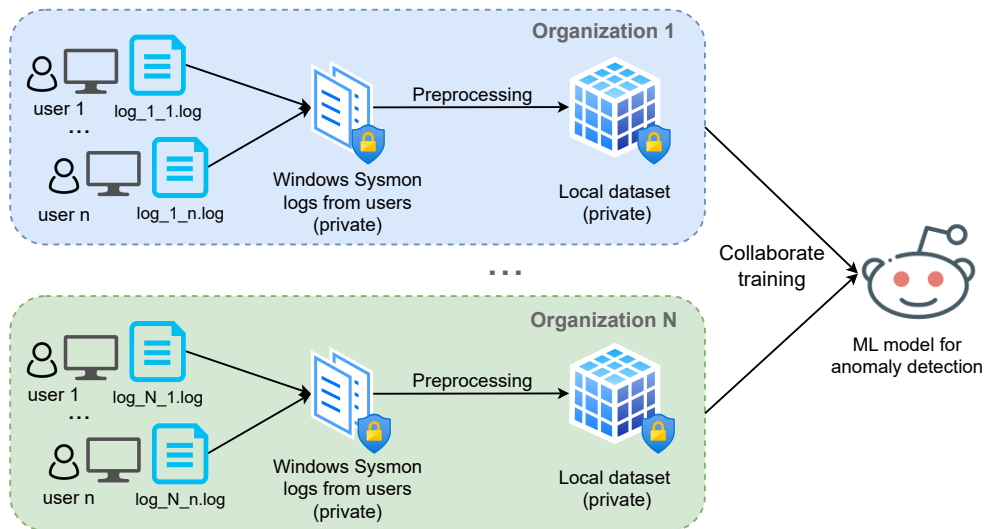


Figure 2.2: An example of threat intelligence sharing.

across the whole system or all organizations. If one organization suffers from the new threat, with TI sharing other organizations have a greater chance to patch it before the vulnerability is exploited [72].

However, TI sharing is facing several barriers that are not only relevant to technology but also policies and company concerns. Considering the example of using Sysmon log data to train the anomaly detection model, organizations will not be allowed to share the raw Sysmon logs with another party since the logs contain sensitive data that organizations do not want to expose. This is a privacy issue that widely exists. Another example that decreases the TI sharing is fearing negative publicity. Some threat information may cause a competitive disadvantage if it is leaked to other parties. And of course, there are many other reasons for not sharing, such as quality issues and untrusted participants [72]. Although FL can be a good alternative for TI sharing [73], companies are still unwilling to share their trained models directly for some concerns. For instance, others might misuse the model without the owner's permission, harming the interests of the model owner. As discussed in Section 2.3.2, attackers are able to extract sensitive information from the model parameters as well, leading to a need of additional security mechanisms.

## 2.6 Related Work

The goal of this project is to protect the data security during the whole lifecycle of FL. There are some other research works and frameworks focusing on

similar topics. In this section, some existing frameworks or techniques are reviewed.

In 2017, V. Bonawitz, et al. [52] proposed one novel failure-robust MPC protocol which leverages Secure Sharing (SS) to handle the arbitrary drop of participants. The security is proved under the host-but-curious and active adversary settings. The idea is to add well-designed masks to the message before sending it to the central server. The sum of all masks is zero thus they don't affect the aggregation result. The key or mask of the dropped user could be retained by Secure Sharing (SS). The paper also designed some optimizing mechanisms for higher efficiency and handling exceptions. The method introduced in this paper is always named as SecAgg. And based on this work, J. Bell et al [75] in 2020 designed a more flexible solution represented as SecAgg+. In SecAgg the masks of each participant are generated by all the other participants, which is not feasible for large-scale scenarios. SecAgg+ improves this by allowing a limited number of masks only shared by the neighbors. One  $(k-1)$ -connected communication graph is used to maintain the state. In this way, the architecture is more flexible and easily implemented for a large number of participants. However, both these two methods require relatively complicated interactions between participants and the central server since the drop of participants in every cycle should be handled, causing extra latency. And it assumes an honest-but-curious server without considering possible malicious insider attacks.

In 2020, K. Wei, et al. [76] designed one differential privacy enabled FL framework called Noising before Model Aggregation FL (NbAFL). Before sending the update to the central server, each participant will first add artificial noise to the model parameters as a mask, so that during the transmission and aggregation, the confidentiality of the local model is enhanced. The global model is also added by noise so that both uploading and downloading transmission are protected. The specific cryptographic primitive is named  $(\epsilon, \delta)$ -DP, where  $\epsilon$  and  $\delta$  are two parameters to control noising, and the noise follows the Gaussian distribution. One distinguishing point is that NbAFL has one theoretical convergence bound on the loss function which could be used to analyze the effect of noising. However, according to the analysis, higher privacy protection results in lower convergence performance, and NbAFL is a better choice when the number of participants is larger. In this project, there are only a limited number of data owners. Thus, NbAFL and differential privacy actually suffer from some limitations.

The following several frameworks are all based on Homomorphic Encryption (HE).

In 2020, C. Zhang, et al. [77] proposed one solution called BatchCrypt that helps to solve the problem of computation and communication costs caused by Homomorphic Encryption in FL. The idea is to quantize the gradients, combining a batch of which into one long integer and encrypting this integer instead of the original gradients. To enable the encrypted batch gradient aggregation, one customized quantization scheme and a new batch encoding scheme are designed. To quantize the gradient values, the clipping technique is necessary. The paper proposed one analytical model called dACIQ to determine the clipping thresholds to minimize the cumulative error. The framework is tested and deployed as one FATE plugin. The experiment shows that it's able to accelerate 23x-93x training speedup and reduce the communication overhead by 66-101x with an accuracy loss of less than 1%. However, although it accelerates the process a lot, the speed is still not fast enough, compared to the speed of processing plaintext.

In 2021, D. Froelicher, et al. [45] designed one decentralized framework called FAMHE that enables distributed analysis collaboration (Federated Analytics) with a strong privacy guarantee, motivated by Multiparty Homomorphic Encryption (MHE). The SIMD parallel computing technique is leveraged to accelerate the computation during the encryption phase. However, the solution may not be feasible enough for general problems, since it uses problem-specific methods to optimize the operations (mainly for GWAS).

In 2021, Z. Jiang, et al. [6] proposed one well-designed HE scheme called FLASHE only for FL scenarios. The paper evaluated the minimum requirements of security and functionality and figured out the reason why existing schemes suffer from heavy communication and computation overhead. Based on their observation, they design FLASHE without asymmetric-key design, and that only supports modular addition operations. Sacrificing the asymmetric-key design and limiting operation, it's possible to achieve one lightweight HE scheme that doesn't induce extra communication cost and only requires slight computation. The paper also designed two variants considering the sparsification demand. With the sacrifice of functionality, FLASHE is able to fulfill the security requirement of Federated Learning with no additional communication overhead. FLASHE is realized as one FATE plugin for evaluation and further implementation. However, as explained in the paper, the design only achieves minimum security requirements, and the flexibility is limited since it only supports additional operations. However, this work could inspire this thesis project and be improved by some additional cryptographic primitives such as TEE.

In 2022, J. Park, et al. [50] proposed one joint FL and HE framework that enables each client to use its own unique key pair. It's achieved by Distributed Homomorphic Cryptosystem (DHC), which uses Multi-Party Computation for implementing distributed homomorphic operations. The core idea is to utilize the DHC to give the distributed Cloud Server (CS) & Computation Provider (CP) system permission to decrypt the client message in a distributed manner. It requires all of the participants to collaborate to decrypt, thus few malicious participants do not matter. The process is well-designed and fully utilizes this DHC mechanism. It utilizes HE to add the noise to the message in CS and let CPs decrypt the noised messages from different clients and calculate sum, then encrypt the result again and send it back to CS. CS removes the noise by HE operation and sends back each encrypted result to the corresponding client. The performance evaluation shows that it's more secure with a sacrifice of computation and communication costs. However, one tricky thing is the assumption of Cloud Server and Computation Provider. The framework requires separate CS and CPs, which is not feasible for small-scale FL, and requires extra communication costs (between CS and CP). Since in this project, the number of participants is limited, it is unnecessary to split control and computation parts, which causes extra cost and system complexity.

In 2022, J. Ma, et al. [78] designed one xMK-CKKS framework that utilizes multikey homomorphic encryption for privacy-preserved federated learning. The framework uses the aggregated public key for encryption and forces all the participants to collaborate for decryption. The evaluation shows that the proposed framework is able to preserve accuracy while enhancing security with lower computational cost compared to Paillier-based FL. This work follows the same idea of designing one multi-key homomorphic encryption scheme, but the architecture is relatively simpler. However, it still suffers from extra communication cost, since it requires extra interaction between clients and central server when calculating the decryption shares.

In 2022, M. Sarhan, et al. [73] designed one FL based TI sharing scheme that allows multiple parties to collaborate in building an ML-based Network Intrusion Detection System (NIDS) in a FL setting. They leverage FL to preserve the privacy during training the model for intrusion detection and compare it with two other scenarios, which are the centralized training without FL and localized training. The experiments are conducted on two key datasets in a NetFlow format. The datasets are NF-UNSW-NB15- v2 and NF-BoT-IoT-v2. The evaluation results show that the accuracy of the federated model is about 90%, only about 8% lower than the centralized model, but significantly better than a localized model, which is only about 52%. However, in this work

the security issues of FL itself are not considered at all.



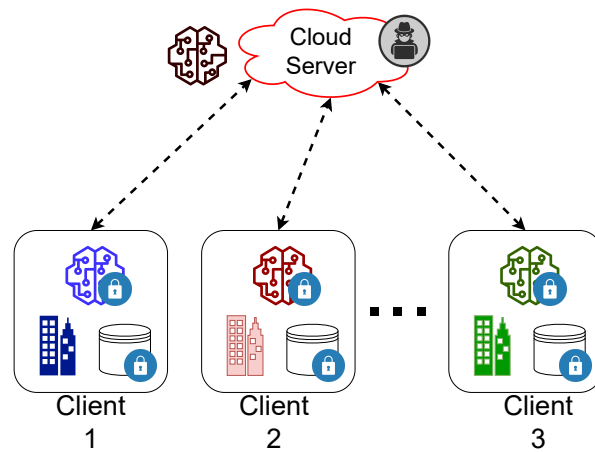
# Chapter 3

## Methodology

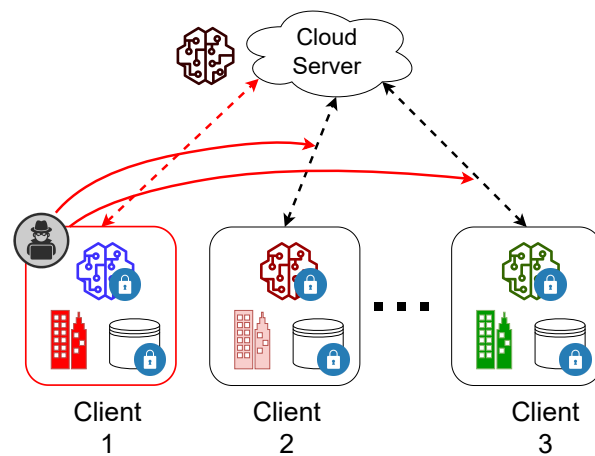
### 3.1 Threat Model

As mentioned in Section 2.3, this project focuses on the HFL scenario, while the total number of participants/parties is finite (smaller than 100), noted as cross-silo federated learning [6]. In this setting, there are four phases that have the risk of information leakage: global model downloading, local training, local model uploading, and model aggregation. In this project, the global model is assumed to be public and not contain any sensitive information. Any party is permitted to access this global model as long as it becomes one participant in the FL process. But if attackers tamper with it during global model downloading, i.e. removing one layer of the global model or modifying some of the model parameters, it might cause a crash of FL process (if the model structure of layers is changed) or a failure of model training convergence (if the model parameters are replaced with other values). Therefore, the global model requires the protection of data integrity in case of malicious manipulation. On the other side, we assume that the local models are private and sensitive. The parameters of one local model are private and contain sensitive information that should be prevented from leaking. Similarly, the violation of the integrity of local models has negative effects on FL training, as the tampered local models will also participate in model aggregation as one kind of poisoning attack [79]. Although there are some advanced approaches [79, 80] that help to defend and mitigate this kind of attack, it is hard to eliminate the effects. Therefore, not only the integrity but also the confidentiality of local models need to be under protection.

To define the threat model for this thesis, we defined two types of adversaries:



(a) Server adversary



(b) Client adversary

Figure 3.1: (a) The server adversary who has root access to the server that provides aggregation service for FL training. (b) The client adversary is honest-but-curious and can eavesdrop on the communication of other benign clients.

1. The adversary is the platform owner that has root access to the cloud platform of CSP, including its storage, registers, running processes and all available resources. Fig. 3.1a illustrates this kind of adversary. It is also able to control all incoming and outgoing traffics. However, the attacks that are based on physical control such as the side-channel attacks like Plundervolt [81] are not considered. This kind of attack violates the system by abusing physical interfaces such as voltage scaling interface, changing firmware behaviors like CPU frequency or voltage and so on, thus it is hard to prevent such kind of attacks unless the manufacturers can optimize their products. This kind of adversary is aggressive and one goal of it is to cheat its users by fake claims or tampered messages, i.e. making the users believe that the model aggregation is going as they expect but actually the server only partially or even does not execute the aggregation. It also aims to extract sensitive information (i.e. local model updates) from the received messages, memory, or running processes.
2. The adversaries participating in the training collaboration (illustrated by Fig. 3.1b). The malicious participants (adversaries) have valid certificates for identity verification and can obtain the keys and configurations for FL training as benign participants. They are able to eavesdrop on the channels between the server and all the other participants. Note that the adversaries are *honest-but-curious*, which means that they follow the agreed FL collaboration without intentionally interfering with it. We assume the adversaries to be honest-but-curious since the adversaries can also get benefits from the FL collaboration. They aim to obtain the local models that contain sensitive information but do not belong to them, violating the data privacy of other parties, without harming their benefits of FL collaboration or leaking their own sensitive data.

Considering the variance of conditions and training environments between the participating parties, it is impossible to provide a general protection approach for their local training and local datasets in this project. Therefore, we assume each party to be responsible for its own data security. This project does not give any special protection against local information leakage that happens on the participant side, i.e. during the local model training. The cooperation of participant adversaries is allowed, which means that multiple malicious participants can cooperate, aiming to obtain the local models of the benign participants. The worst case is that most but one participant is

malicious. However, the cooperation of different types of adversaries is out of scope for this project, and it is reasonable since this kind of cooperation will harm the FL training itself, thus violating the assumption of the second kind of adversary.

## 3.2 Framework Developing Methods

This section introduces the steps of developing the framework and tools that are going to be used.

### 3.2.1 Startup

The work starts on the experimental environment setup. Before deploying the prototype to TEE hardware, the framework is developed and simply tested on a VM instance. To simulate the FL process, one virtual distributed system on a single machine will be leveraged to do experiments.

As mentioned in Section 2.3.3, this project will be based on Flower framework [82]. Flower is one Python library that supports creating FL server process and client process separately. Thus, to build the environment for experiments, firstly we use pipenv [83] to create the virtual Python environment and install all the dependencies inside it. One benefit is that pipenv will record all the dependencies in one configure file, making it easy to reconstruct the same environment on another machine in the future.

After installing the Flower library and all the required dependencies as well, the next step is to set up the FL training process. We first start with Pytorch, with which we are the most familiar. According to the tutorials given by Flower, it is easy to build the server and multiple clients following the steps:

1. Define the training and testing function in the client script.
2. Define the client class, specify the training process in the *fit* inner method and the testing process in the *evaluate* inner method.
3. Define the server class in the server script, using build-in strategies.
4. Run the server in one terminal and run at least two clients in separate terminals. Then the clients will communicate with the server and start FL training.

### 3.2.2 Implementing HE

To work efficiently, we plan to do our work on top of existing extensions. As mentioned in Section 2.3.3, there are some existing HE extensions that could be combined into the Flower framework easily, just by defining some new clients and server strategies. However, considering that different HE schemes have different limitations, i.e. CKKS allows encryption on float values directly while BFV only supports encryption of integers, some additional functions, such as quantization or data type converting, are also necessary. And because each HE library only supports 1 or 2 schemes, it might also need to apply more than one single extension. In this project, we plan to first apply 1 or 2 extensions to support at least 3 different HE schemes and combine them into the Flower FL training process.

Although FLASHE extension [59] of FATE is not complete and lacks the necessary documents, the idea is brilliant and seems not very different to adjust it to Flower, we also plan to implement it into Flower as a new extension. However, it might require us to dive into the source code of the FLASHE extension to extract the core code. BatchCrypt [11] is proved to be one great approach to accelerate the computation and reduce the size of encrypted messages for existing HE schemes. If time allows, we also plan to implement it to see how much it helps to improve efficiency.

### 3.2.3 Preliminary Tests

Before testing the framework with TI task, it will first be evaluated with some preliminary tests, which rely on the build-in datasets provided by Flower framework for testing. These datasets are relatively small and do not take much time to complete. The purpose is to check if the framework is able to complete the FL process as expected and if the HE function works well. It is necessary to optimize the framework until it passes all the preliminary tests.

## 3.3 Framework Testing

After developing the framework and before deploying it to TEE equipment. We plan to conduct the tests to figure out its performance only with HE on the TI problem. The planned test methods are as follows.

### 3.3.1 Planned Test Settings

After developing the framework, the next step is to evaluate its security performance and efficiency. Although this framework aims to fulfill general FL tasks, in this project it will only be leveraged to train the model for the TI problem. The test scenario is set as a HFL where organizations or enterprises share the information for better collaboration and one central server performs as the aggregator. Under this scenario, the tests will consider different FL settings. This project plans to mainly consider three aspects:

1. **ML models:** The sizes of different models are diverse, thus the efficiency of computation and communication may be affected as it takes more time to encrypt large models and the encrypted message might be larger.
2. **FL scale:** The number of clients will significantly impact the model performance and efficiency of HE schemes because the number of homomorphic operations increases when more clients participate in the training.
3. **HE schemes (if available):** By using different HE schemes, the time cost of encryption/decryption varies a lot. And the sizes of encrypted messages are also significantly different, causing a difference in communication cost.

### 3.3.2 Evaluation Metrics

To evaluate the efficiency and practicality of the framework, this project is going to consider several metrics:

1. **Security Assessment:** It is necessary to assess whether the framework has the ability to defend the threat models defined in Section 3.1.
2. **Model Accuracy:** The encryption/decryption of HE might add noise to the model parameters, which could have negative effects on model performance.
3. **Time cost:** Since additional security features cause computational overheads, it is necessary to measure the extra time these features take.
4. **Traffic Overhead:** After encrypting by HE, the encrypted message might be larger than the plaintext message, causing a higher communication

cost when transmitting it. One efficient way to evaluate this extra communication cost is to measure the increase in the message size directly.

### 3.3.3 Datasets

The dataset we used in this project is one TI dataset provided by CanaryBit and one MNIST dataset [84] for the classification of handwriting numbers. By testing the framework on these two different kinds of datasets, the evaluation results are more reliable and convincing.

#### 3.3.3.1 TI dataset

The TI dataset includes three sub-datasets that consist of logs from three different processes: *cmd.exe*, *powershell.exe* and *winword.exe*. The logs are collected from multiple Windows System devices and combined into these three sub-datasets. To obtain the dataset from raw log files, there are several steps:

1. Parse the raw log files into CSV tables.
2. Combine all the CSV files into one single file and do the preprocessing phase 1, including path normalization and anonymization.
3. Do preprocessing phase 2, including categorization and converting to numerical values.

After preprocessing the data, we got the dataset that is ready for training ML models.

#### 3.3.3.2 MNIST dataset

The MNIST dataset [84] is a widely-used ML dataset, which consists of a total of 70,000 gray-scale images (60000 for training, 10000 for testing) of handwritten digits from 0 to 9. It is designed for training and testing general ML models and serves as a benchmark dataset specifically for image classification tasks. What is more, prevalent ML frameworks like Pytorch or Tensorflow already have built-in support for this dataset, making it convenient to run simple tests on it.

## 3.4 Prototype Deployment

The resulting prototype will be deployed and tested in AMD SEV hardware. In our project, the implementation will rely on x86 platforms accessed using AWS platform\*.

After deploying, we are going to conduct the evaluation and compare its performance with the results gained before.

---

\*AWS SEV-SNP: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html>

# Chapter 4

## Framework Design

This chapter includes a detailed description of system design, improvements, and algorithms. As the framework combined HE and FL, and is realized as a plugin on Flower framework [8], the designed framework is named as Homomorphic Encryption Federated Learning FLower Plugin (Heflp).

The rest of this chapter is organized as follows. Firstly the system overview and the FL process are introduced in Section 4.1 and 4.2. Section 4.3 explains the supported HE schemes and the modified Flashe scheme *Flashev2*. Section 4.4 introduces the quantization and weighted averaging strategy required for some HE schemes.

### 4.1 System Overview

The system adopts a client-server architecture (see Fig. 4.1). The client side consists of multiple client instances each operated by a distinct contributing party. The server side only represents one server application that runs on the cloud platform and provides the aggregation service for FL. Each client maintains one TLS connection with the server and transmits data on it (explained in Section 2.4).

Each client runs two core components (underlined in Fig. 4.1). The local training component plays the role of training the received model on the private local dataset. In this project, we assume that the local training is secure (it will not leak sensitive data) and the local dataset can only be accessed for local training. The threat of local information leakage is out of scope and can not be prevented by this framework. After training, the local model is forwarded to the HE cipher component. The cipher encrypts the received model from local training based on a certain HE scheme and then transmits the encrypted data

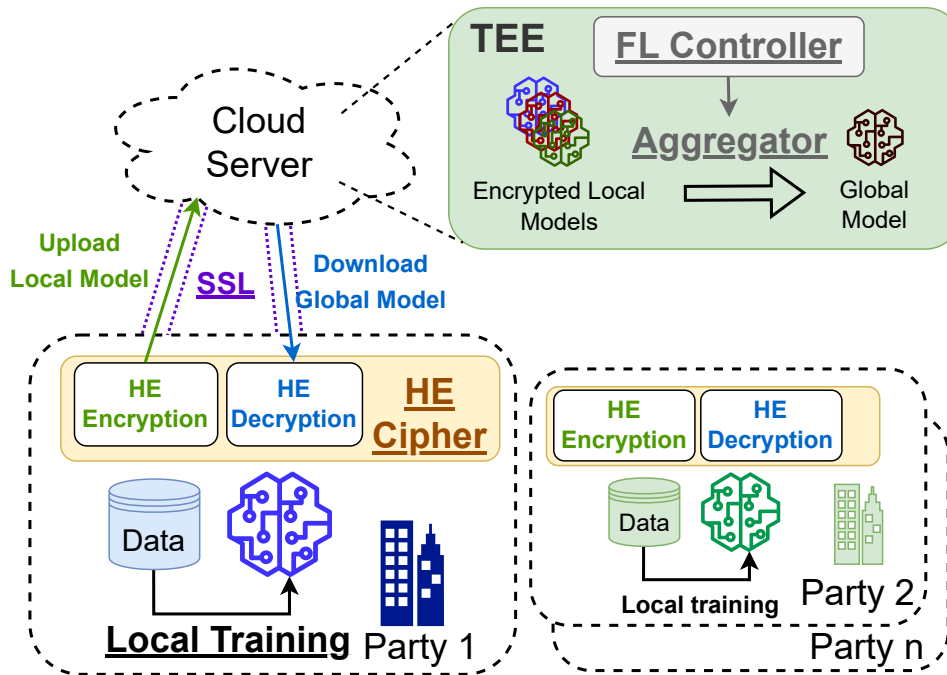


Figure 4.1: System overview

to the server. Accordingly, it is also responsible for decrypting the received data from the server using the same HE configuration, then forwarding the decrypted model parameters to the local training component to trigger the next round of training.

The server includes one aggregator for aggregating the received model weights and one FL controller to manage the FL and HE configuration (underlined in Fig. 4.1). The aggregator receives the encrypted model updates, aggregates them without learning any sensitive information, and sends the aggregated result back. The FL controller decides the configuration parameters that need to be sent to clients, e.g. number of training epochs, training setting, and context of HE cipher (see more details in Section. 5.1.4). Another important role of FL controller is to manage FL training, i.e. instructing the aggregator to use the correct function for specific HE scheme, collecting and analyzing the evaluation results and deciding if need to continue training or stop. Both the aggregator and the FL controller are running inside one TEE to protect the integrity of data and process, which means that any outside adversary can not manipulate the aggregation process or modify the input and output of aggregation.

## 4.2 Federated Learning Process

In this project, the FL process follows the classic HFL pattern explained in Section 2.3, and enhances the security via additional HE mechanisms. The designed framework supports multiple HE schemes but follows a similar procedure. As shown in Fig. 4.2, the process consists of three stages: the initial setup, FL training and completion.

### 4.2.1 Stage 1: Initial Setup

The first stage is the initial setup of the framework, including the initialization of HE cipher and quantizer in each client, setup of clients and the server, and building the communication channels between clients and the server. In this stage, the FL controller inside the server also loads an initial model and generates the corresponding configurations (explained in Section 4.1), then sends them to all the clients and waits until receiving a predetermined number of responses from clients. Each client installs the received initial model and sets up the local training environment according to the received configuration parameters.

### 4.2.2 Stage 2: FL Training

The second stage is the main part of FL process. In this stage, all clients collaborate on training the global model iteratively with the help of the aggregation server until achieving the convergence of model training or the maximum number of rounds preset in the initial setup. In each round, each client first trains the model on its local dataset. Then all local models are transmitted to the server after being quantized (if required by the HE scheme) and encrypted. Considering that when using specific HE schemes clients may also need to send some additional data (see more details in Section. 5.1.3), e.g. quantization parameters and the range of model weights, an optional field called metadata is attached as well. After receiving enough updates from clients, the server aggregates all the received local models and updates the configuration for each client, then sends back the updated global model to clients with corresponding configurations. On the client side, each client is able to obtain the global model by decoding the received message with the same quantizer and cipher. Before moving to the next round of training, the server requires a predetermined number of clients to evaluate the performance of the updated global model and send back the evaluation results so that

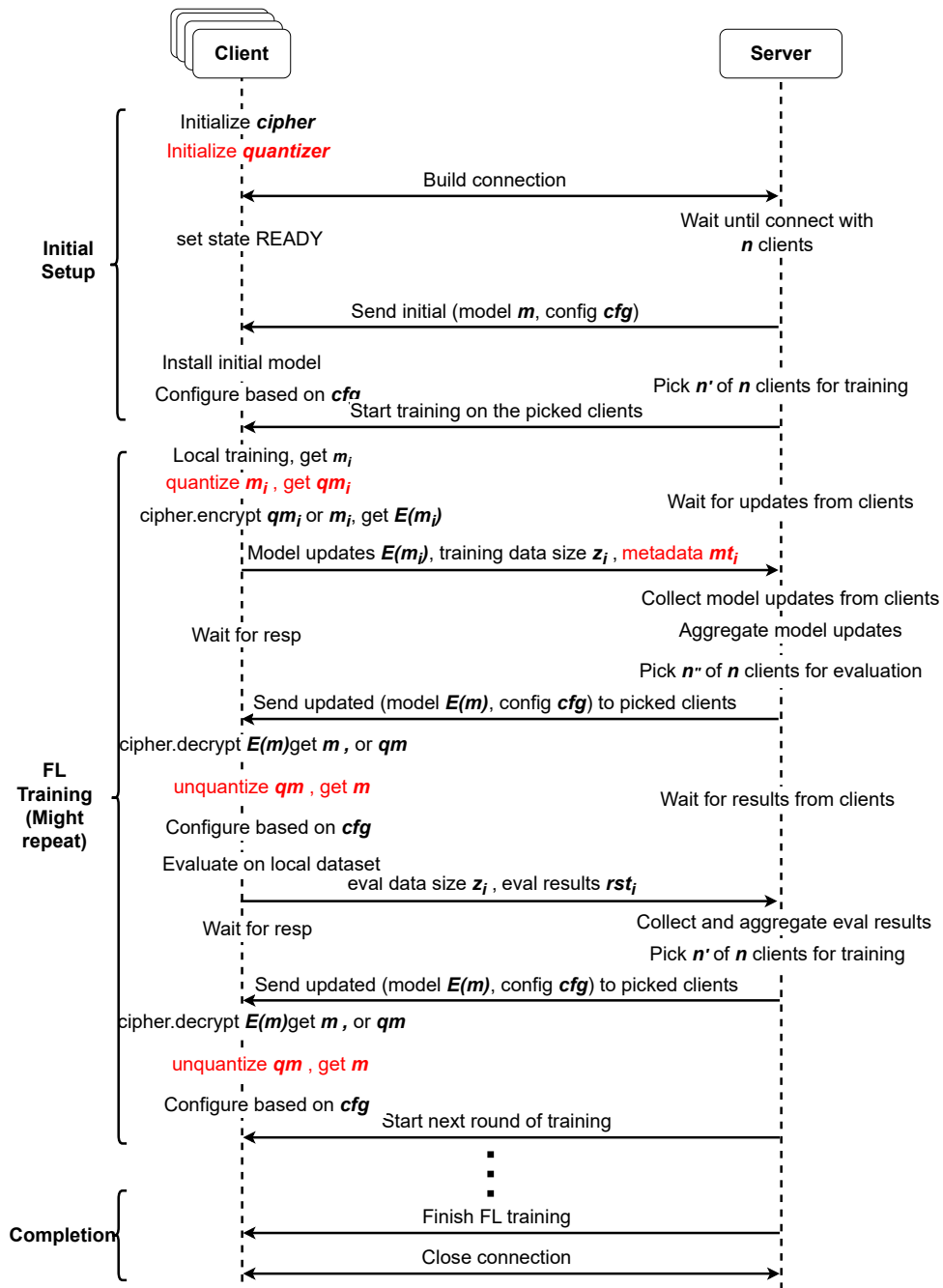


Figure 4.2: Federated Training process with Homomorphic Encryption, red marks mean that these steps are optional and only necessary for some of the HE schemes

it can collect, record, and analyze such kind of information for monitoring the training process and making better decision whether to stop training earlier. The evaluation results can also be helpful for other purposes such as setting better training configurations and evaluating the effects of different HE schemes and FL algorithms on convergence speed or global accuracy. However, it always takes additional time and the users need to evaluate the trade-off.

### 4.2.3 Stage 3: Completion

The last stage is to complete training. When the global model is converged or the training reaches the maximum number of rounds, the server stops and instructs clients to end the FL process as well.

## 4.3 Homomorphic Encryption Schemes

This section mainly introduces the HE schemes that this framework implements. As shown in Table 4.1, there are totally 4 HE schemes implemented in Heflp: *Flashe*, *Flashev2*, *CKKS*, *BFV*. All these schemes have different characteristics and fit different scenarios.

Table 4.1: Supported homomorphic encryption schemes

Scheme	Data Type	Operation	Quantization	Dependency
Flashe[6]	integer	+	Require	Numpy
Flashev2	integer	+, ×L	Require	Numpy
CKKS[28]	float	+, ×	Not require	Pyfhel[63]
BFV[27]	integer	+, ×	Require	Pyfhel

Note: L means constant in Flashev2

### 4.3.1 Flashe

*Flashe* [6] follows the idea to design one HE scheme specifically for cross-silo FL. Considering that current prevalent FL algorithms like FedAvg [1] or FedProx [47] only require a weighted average operation, there is no demand for supporting multiplicative operation for HE in FL. In FL, it is only necessary for HE scheme to support addition. Because both encryption and decryption happen only on the client side, the asymmetric design is also redundant. Therefore, in this context, symmetric design is a more efficient solution,

which uses only one key for both encryption and decryption instead of a key pair (private key and public key) for encryption and decryption respectively. Therefore, *Flashe* is designed as a symmetric HE scheme that only supports additive homomorphism to accelerate the encryption speed and minimize the size of the ciphertext. And thanks to the quantization technology, which is able to rescale a range of real numbers to a range of integers, *Flashe* is allowed to only support integer encryption. As shown in Fig. 4.3, the idea is to add masks of random numbers to the plaintext and record the parameters used for generating masks during calculation.

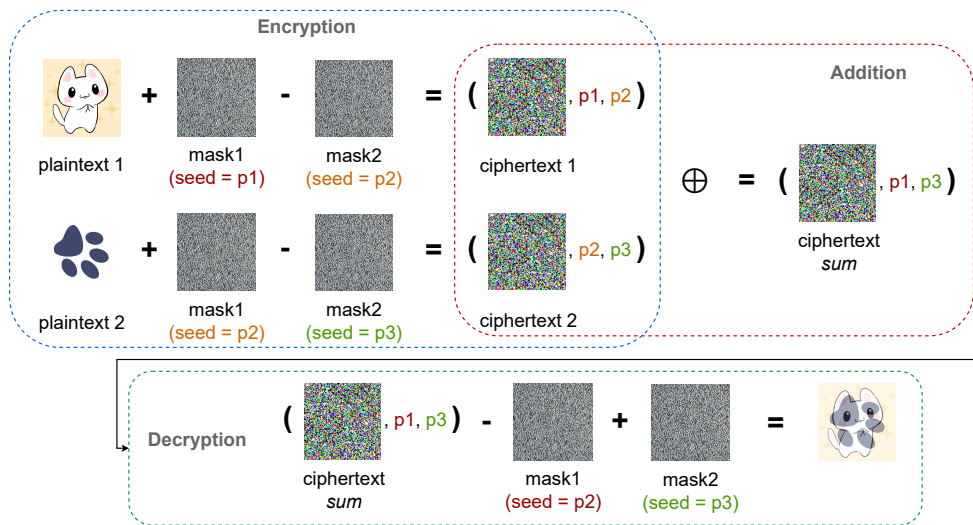


Figure 4.3: An example of *Flashe*. Use *Flashe* to calculate the sum of two pictures securely.

The core cryptosystem consists of encryption and decryption processes. The encryption process is defined as:

$$E_k(m) = (c, i, S : \{j\}),$$

$$s.t. \quad c_d = (m_d + F_k(i \parallel j \parallel d) - F_k(i \parallel (j + 1) \parallel d)), \quad (4.1)$$

$$for \quad 1 \leq d \leq D$$

where  $m \in \mathbb{Z}_n^D$  is the plaintext that needs encryption and  $D$  represent the length of it.  $n$  is the modulus.  $i, j$  are two parameters to generate random masks. In FL  $i$  can be the number of rounds while  $j$  is the client id.  $k$  means the key and  $F_k : I \rightarrow \mathbb{Z}_n$  represents a Pseudorandom Function (PRF) that uses  $k$  as the key to map the given value to a random value as a mask.

The additive operation is defined as:

$$(c_1, i, S_1) \oplus (c_2, i, S_2) = (c_1 + c_2, i, S_1 \cup S_2) \quad (4.2)$$

where the addition consists of two steps: do the modular addition on  $c_1$  and  $c_2$ , and combine the two  $S$  sets together.

Accordingly, the decryption is the process of recreating the masks according to the recorded  $S$  set and removing them to get the decrypted result:

$$\begin{aligned} D_k((c, i, S : \{j_1, j_2 \dots\})) &= m, \\ \text{s.t. } m_d &= (c_d + \sum_{j \in S} (F_k(i \parallel (j + 1) \parallel d) - F_k(i \parallel j \parallel d))), \\ &\text{for } 1 \leq d \leq D \end{aligned} \quad (4.3)$$

During decryption, as all the masks should be calculated and removed from the ciphertext to recover the decrypted result, the calculative overhead increases with the size of  $S$ . So the time complexity is  $O(ND)$  where  $N$  is the number of local models that participate in aggregation. One design in *Flashe* to reduce such kind of overhead is to carefully generate two masks when encryption using  $j$  and  $j + 1$ . In this way when adding up the encrypted model parameters the masks can be canceled by each other, and ideally the overhead could be reduced to  $O(D)$ .

When initializing a *Flashe* cipher, two parameters are required: the key  $k$  for PRF generating the masks and the bit-width that defines the maximum range of value that *Flashe* can protect, i.e. *Flashe* can only hide the values in range  $[-2^{r_e-1}, 2^{r_e-1}]$  if bit-width is denoted by  $r_e$ .

### 4.3.2 Flashev2

Although *Flashe* is a good HE scheme choice specifically for FL, it still suffers from some shortcomings:

1. As the rule to set  $i$  and  $j$  is public and all the clients use the same key for both encryption and decryption if one malicious user is able to eavesdrop on and obtain the encrypted models from other benign users, it can recover the plaintext by using its key. So *Flashe* scheme is vulnerable to such an inside attack. We expect to hide  $i$  and  $j$  from other clients to enhance the data privacy between clients.

2. *Flashe* only supports additive operation and does not allow the multiplication between ciphertext and a constant. Therefore, the strategy of calculating a weighted average of model parameters relies on a trick to calculate the multiplication of plaintext and its weight, as well as the division of the sum of results and weights locally. However, this will cause a problem of additional computational overhead or a loss of accuracy for the clients. More details will be explained in Section 4.4 later. It could be more flexible to expand the ability of *Flashe* with a feature to support the multiplicative operation of ciphertext and constant.

To address these problems, we optimized and modified the cryptosystem of *Flashe*. The optimizations rely on two ideas: 1. Replace the  $i$  and  $j$  with a customized field  $p$  and let the server define the  $S : \{p_1, p_2 \dots\}$  for encryption so that the  $S$  remains private to enhance the privacy between clients. 2. Attach one  $t$  parameter to each item  $p$  in  $S : \{(p_1, t_1), (p_2, t_2) \dots\}$ , which specifies the multiplier when adding the masks to plaintext or removing them from the ciphertext. The revised version of *Flashe* is named as *Flashev2*. Similarly, we give another example Fig. 4.4 to show the process of *Flashev2*.

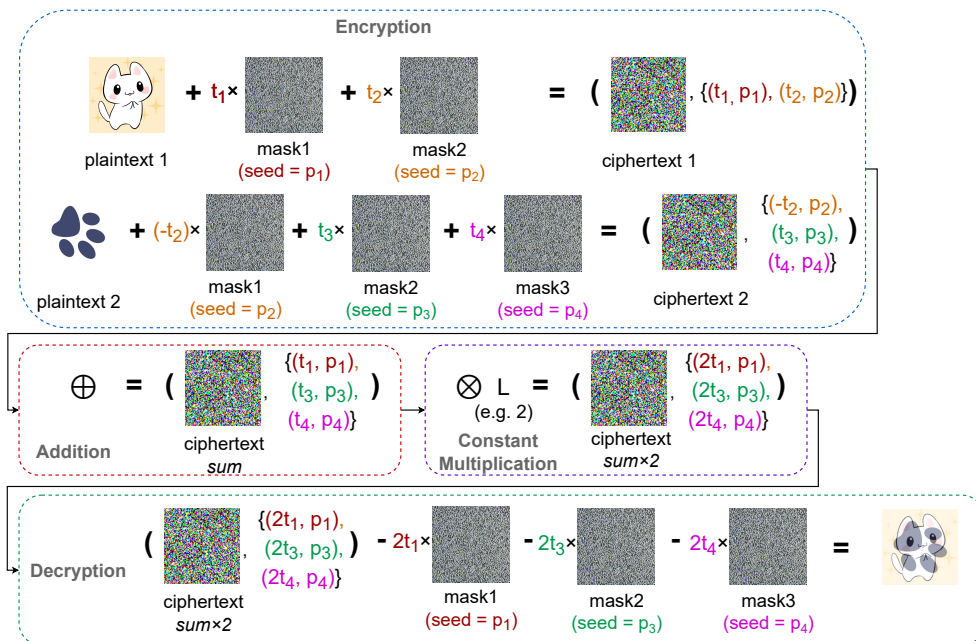


Figure 4.4: An example of *Flashev2*. Use *Flashev2* to calculate the sum of two pictures securely.

For the encryption, the definition of encrypted result is modified as:

$$\begin{aligned}
 E_k(m) &= (c, S : \{(p_1, t_1), (p_2, t_2) \dots\}), \\
 \text{s.t. } c_d &= (m_d + \sum_{p,t \in S} t \cdot F_k(p \parallel d)), \\
 &\text{for } 1 \leq d \leq D
 \end{aligned} \tag{4.4}$$

During encryption with *Flashev2*, the generation of masks is fully decided by the  $S$ , which is specified by the server and kept private for each client. Currently, we use two shuffled lists of client id as  $p_1$  and  $p_2$  for each client, e.g.  $[2, 1, 4, 3]$  and  $[4, 2, 3, 1]$  when having 4 clients. To retain the advantage of mask self-canceling,  $t_1$  is set as 1 while  $t_2$  is  $-1$ . However, the server can apply an arbitrary strategy to generate this  $S$  for each client, i.e. using a different method to decide  $p$  and  $t$  or more  $p, t$  pairs. As each mask is added to the plaintext with a multiplier  $t$ , if we set  $p_1 = i \parallel j, t_1 = 1$  while  $p_2 = i \parallel (j + 1), t_2 = -1$ , *Flashev2* will behave the same as *Flashe*.

By adding the  $t$  field as part of each item in  $S$ , *Flashev2* enables the multiplicative operation between ciphertext and a constant. The operation is defined as:

$$(c, S) \otimes L = (c \cdot L, S \cdot t \cdot L) \tag{4.5}$$

where the multiplicative operation consists of two steps: do the modular multiplication on  $c_1$  and constant  $L$ , and multiply every  $t$  in  $S$  with  $L$ . The additive operation of ciphertext is the same as Equation 4.2.

The decryption is very similar to the encryption where the only difference is reversing  $t$  to remove the masks:

$$\begin{aligned}
 D_k((c, S : \{(p_1, t_1), (p_2, t_2) \dots\})) &= m, \\
 \text{s.t. } m_d &= (c_d - \sum_{p,t \in S} t \cdot F_k(p \parallel d)), \\
 &\text{for } 1 \leq d \leq D
 \end{aligned} \tag{4.6}$$

*Flashev2* has the same parameter settings as *Flashe*, comprising the key  $k$  and bit-width  $r_e$ .

### 4.3.3 CKKS & BFV

The *CKKS* scheme [28] is one variant of HE schemes that specifically work on real and complex numbers. It only supports approximate arithmetic operations (both additive and multiplicative operations) on encrypted data. So it will give approximate results of homomorphic operations instead of exact results. The level of precision is decided by some parameters e.g. polynomial modulus degree. These parameters impact the computational efficiency as well.

The *BFV* [27] scheme is another popular HE scheme, which supports the encryption on integer plaintexts. Unlike *CKKS*, *BFV* is able to provide exact arithmetic operations on encrypted data. There are several parameters like polynomial modulus degree and plaintext modulus that can trade-off between precision tolerance, security level, and computational efficiency.

Both *CKKS* and *BFV* are FHE schemes that support arbitrary arithmetic operations on their allowed data types. However, in FL scenario, as the weights of ML models are real numbers, specifically for *BFV* it requires first quantizing them to integers before performing encryption.

This report will not discuss the details of principles of *CKKS* and *BFV* schemes because we mainly care about their application and performance in the FL scenario and did not do any modification to these two HE schemes.

## 4.4 Quantization and Weighted Averaging

As discussed before, considering that some HE schemes (*BFV*, *Flashe* and *Flashev2*) require converting real model weights to integer values before encryption, it is necessary to implement an appropriate quantization method. On the other hand, the encrypted ciphertext does not support arbitrary divisive operation. However, the aggregation needs to perform the weighted averaging operation on the parameters of local models. Therefore, we need to design one weighted averaging strategy to realize such an operation. The quantization and weighted averaging can be mixed together as one calculation strategy, so both of them are discussed in this section.

### 4.4.1 Symmetric Quantization

The classic  $r$ -bit quantization function  $Q$  for range  $[x_{min}, x_{max}]$  and its corresponding reverse function are defined as:

$$\begin{aligned}
 Q(x) &= \lceil \lceil (2^r - 1) \cdot \frac{(x - x_{min})}{(x_{max} - x_{min})} \rceil \rceil \\
 Q^{-1}(q_r) &= q_r \cdot \frac{(x_{max} - x_{min})}{2^r - 1} + x_{min}
 \end{aligned} \tag{4.7}$$

where the  $\lceil \cdot \rceil$  denotes the rounding function. However, it can not be applied to the FL process directly because of two weaknesses:

1. Because of the  $+x_{min}$  item in the reverse quantization (or dequantization) function  $Q^{-1}$ , when adding  $n$  quantized values up, the dequantization function has to convert to  $Q^{-1}(q_r) = q_r \cdot \frac{(x_{max} - x_{min})}{2^r} + x_{min} \cdot n$ , making things complicated as the server need to record the number  $n$ .
2. The quantized value is always larger than 0, so the accumulative result is easy to overflow during aggregation, especially when the number of FL clients is large. For example, suppose each client only sends one model parameter to the server for aggregation, when using the 16-bit quantization, the range of quantized value  $q$  is  $[0, 2^{16} - 1]$ . If total  $2^6 = 64$  clients participate in aggregation and the weight of every client is set as  $2^{10} = 1024$ , the weighted additive result can be expressed as  $sum = \sum_{i=1}^{64} q_i \cdot 1024$ , thus the range of  $sum$  is expanded to  $[0, 2^{32} - 1]$ . If the signed integer type is 4 bytes (32 bits), this sum might cause an overflow problem, which means that the sum result is larger than the maximum value one integer variable can represent, not to mention the use of HE scheme that could make the available range narrower. Once the overflow occurs, it has to restart the calculation to avoid this problem and cause extra time overhead.

To mitigate these problems, we first turned to the symmetric quantization method that is used in BatchCrypt [77]. The idea is simple, instead of quantizing any value into a positive integer, symmetric quantization maps  $[-\alpha, 0]$  and  $[0, \alpha]$  to  $[-(2^{r-1} - 1), 0]$  and  $[0, 2^{r-1} - 1]$  separately, so that the range of quantized values is symmetric. The mapping function is the same as Equation 4.7 where  $x_{min} = 0$  and  $x_{max} = \alpha$ :

$$\begin{aligned}
 Q_s(x) &= Sign(x) \cdot \lceil \lceil (2^{r-1} - 1) \cdot \frac{|x|}{\alpha} \rceil \rceil \\
 Q_s^{-1}(q_r) &= q_r \cdot \frac{\alpha}{2^{r-1} - 1}
 \end{aligned} \tag{4.8}$$

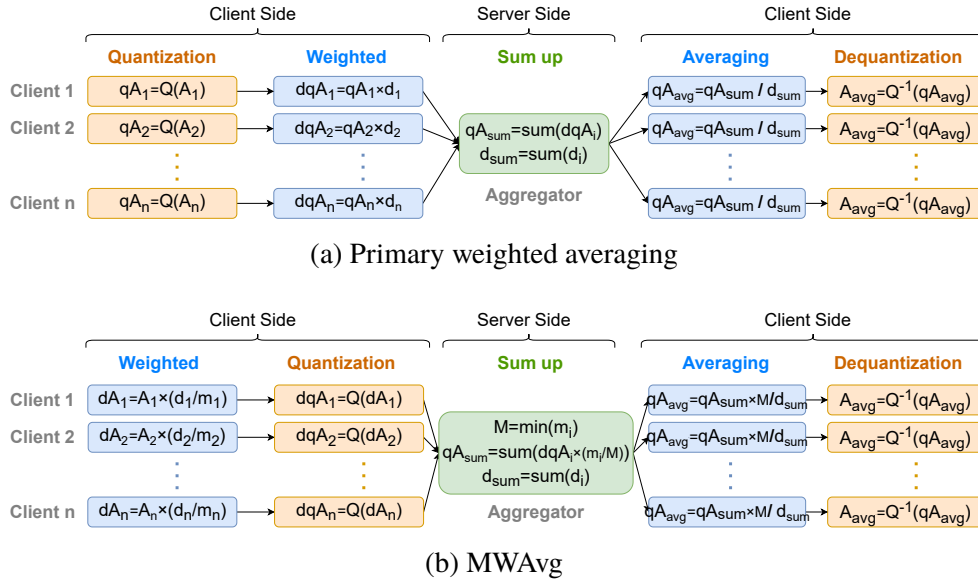


Figure 4.5: (a) Working flow of the primary weighted averaging process.  $A_n$  denotes model parameter vector of client  $n$  and  $d_n$  denotes the weight of client  $n$ . The total number of clients is  $n$ . (b) Working flow of the proposed *MWAvg* process, following the definition given by Equation 4.10.

There are several advantages of symmetric quantization compared with the standard one. Because the  $x_{min} = 0$ , the problem caused by the  $+x_{min}$  item is solved when calculating the sum. Another main property is that the values with opposite signs can cancel each other during the aggregation, making it more robust to overflowing than the standard quantization. However, only symmetric quantization is not enough when calculating the weighted averaging result. The following section introduces strategies for weighted averaging and the challenges.

Recent work shows that it is enough to use 16-bit variables for quantizing model parameters to retain the accuracy [85]. Therefore, in this project, we constantly set  $r=16$ .

#### 4.4.2 Primary Weighted Averaging Strategy

As discussed before, for those HE schemes that do not support encryption and operation on real numbers, the primary weighted averaging process can be divided into three steps: 1. Multiply the quantized local model parameters of each client with its weight on the client side; 2. Aggregate all the local models by adding up their parameters on the server side; 3. Divide the sum result by

the sum of weights on the client side and then dequantize to get the final result. The working flow is shown as Fig. 4.5a.

The reason to quantize the model parameters before multiplying with weight is that quantization requires the input value to be within the same range  $[-\alpha, \alpha]$  for any client no matter how much weight it performs. The boundary value  $\alpha$  has to be the same among all the clients to guarantee the correctness of model aggregation. For each client, as the weight is arbitrary, if firstly multiplying the model parameters with weight, all the clients need to negotiate the  $\alpha$  value before quantization, making it much more complicated and increasing the communication overhead. However, quantization before weighting increases the possibility of overflowing as the weight can be an arbitrary value and frequently change. Suppose  $r = 2^{16}$  of quantization and the weight of one client is the size of the local dataset, which could be much larger than  $2^{10} = 1024$ . In this case, the  $d_{sum}$  can be larger than  $2^{16}$  with a high possibility (if client number  $n \leq 2^6$ ), making the  $qA_{sum}$  easy to overflow (if the maximum value is  $2^{31}$ ). Therefore, one challenge of weight averaging is to handle the arbitrary weights and avoid overflowing when calculating the sum of model parameters. Another challenge is addressed in one possible case that weight values are not integers but real numbers.

### 4.4.3 Improved Weighted Averaging Strategy: MWAvg

To address the challenges, we proposed one advanced weighted averaging strategy by rescaling its range as early as possible. The proposed strategy is named as *MW*Avg.

Considering the formula of weighted averaging is:

$$q_{avg} = \frac{\sum_{i=1}^n q_i d_i}{\sum_{i=1}^n d_i} \quad (4.9)$$

where  $q_i$  is the value of client  $i$  that participate in weighted averaging and the  $q_{avg}$  is the averaging result. After adding some items to it, we can divide the  $q_{avg}$  into the multiplication of several items:

$$q_{avg} = \frac{\sum_{i=1}^n q_i d_i}{\sum_{i=1}^n d_i} = \left( \sum_{i=1}^n q_i \cdot \frac{d_i}{m_i} \cdot \frac{m_i}{M} \right) \cdot \frac{M}{\sum_{i=1}^n d_i} \quad (4.10)$$

where  $m_i = 2^{\lceil \log_2(d_i) \rceil} \in [d_i, 2d_i]$ ,  $M = \min(m_i)$

The idea is to first rescale the weight  $d_i$  to range  $[0, 1]$  so that the model

parameters can be multiplied with it before quantization, keeping the same  $\alpha$ . As shown in Equation 4.10, we transform weight  $d_i$  into  $\frac{d_i}{m_i}$ . Because  $m_i = 2^{\lceil \log_2(d_i) \rceil} \in [d_i, 2d_i]$ ,  $\frac{d_i}{m_i} \in (0.5, 1]$ , fits our requirement. Considering that the  $m_i$  can be different among various clients, we need one mechanism to generalize it to get the correct result. To achieve this, we modified the aggregation process on the server side. Instead of simply adding all the received vectors  $dqA_i$  up, the server first selects the minimum value among all  $m_i$  as  $M$ , then calculate the weighted sum of  $dqA_i \cdot \frac{m_i}{M}$  where  $\frac{m_i}{M}$  is the weight. Every  $m_i$  is a power of 2, thus  $\frac{m_i}{M} \in N$  and the integer multiplication is legal. Finally, multiply the sum with the last item  $\frac{M}{\sum_{i=1}^n d_i}$  to get the weighted averaging result on the client side. According to the Equation 4.10, the final result should be equal to the result of weighted averaging. The working flow of *MWAvg* is shown in Fig. 4.5b.

*MWAvg* is designed to handle arbitrary weights when aggregating models using weighted averaging. It splits the weight  $d_i$  into two parts  $\frac{d_i}{m_i}$  and  $\frac{m_i}{M}$  and apply them in different steps. As discussed before, the main negative effects caused by arbitrary weights are overflowing and difficulty in handling real-number weights. By using *MWAvg* these negative effects are significantly absorbed by  $m_i$  so that it will not impact the aggregation much. Suppose the maximum  $d_1 = 100000$  and the minimum  $d_0 = 1000$ ,  $m_1 = 2^{17}$  and  $m_0 = M = 2^{10}$ , respectively. In this case,  $\frac{m_1}{M} = 2^7 = 128$  and  $\frac{m_0}{M} = 1$ , ensuring the safety of aggregation against overflowing even when the number of clients is large, i.e. over 100 clients. Intuitively, the second part of weight  $\frac{m_i}{M}$  is approximately the division of  $d_i$  and minimum  $d$ . If it is larger than 100, it means the weights are not set reasonably as some clients use significantly smaller weights than others and contribute too little to the FL training. On the other hand, *MWAvg* works well for real weights. For float weights, *MWAvg* works the same as for integer weights, with no need for any other transformation. However, because the original values are multiplied by  $\frac{d_i}{m_i} \in (0.5, 1]$  before quantization, it will result in a maximum 2 times higher error of quantization if  $\frac{d_i}{m_i} \rightarrow 0.5$  compared to the primary weighted averaging.

#### 4.4.4 Flashev2+MWAvg

For primary weighted averaging, the HE is applied after quantization and weighting so that the weight can be multiplied to the plaintext before encryption. This working flow works fine for CKKS and BFV. However, it might weaken the security provided by *Flashe* and *Flashev2*. Suppose for

client  $i$ , the quantized model parameters  $qA_i$  are 16-bit integers and the weight is  $d_i = 2^6 = 64$ , *Flashe* can directly generate and add 16-bit masks to the plaintext without weighting. However, after multiplying the quantized values with the weight  $d_i = 2^6$ , the range  $[-(2^{15} - 1), 2^{15} - 1]$  is also expanded to  $[-(2^{15} - 1)2^6, (2^{15} - 1)2^6]$ . If continue using the 16-bit masks, only the lower 16 bits are protected while the other 6 bits are still plaintext, causing a security issue. Although it can be fixed by generating masks of more bits, i.e. the masks should be able to cover the possible range of plaintext values  $[-(2^{15} - 1)2^6, (2^{15} - 1)2^6]$ , one valid mask will consume at least 21 bits. It is also very hard to decide how many bits should be used because the weight is arbitrary and results in larger memory and computational overhead.

Another possible solution is to encrypt before weighting if the weight is integer, *Flashe* can use the same 16-bit masks. For *Flashe*, the multiplication of ciphertext and integer weight  $c \cdot d$  (*Flashev2* supports this operation but *Flashe* does not) can be transformed as the sum of many ciphertext  $\sum_1^d c$ . But this will break the self-canceling property of *Flashe* during aggregation as long as the weights are not exactly the same, significantly increasing the overhead of decryption as all the accumulated masks need to be removed. In this case, the more clients that participate in aggregation, the more time decryption consumes to remove all the masks.

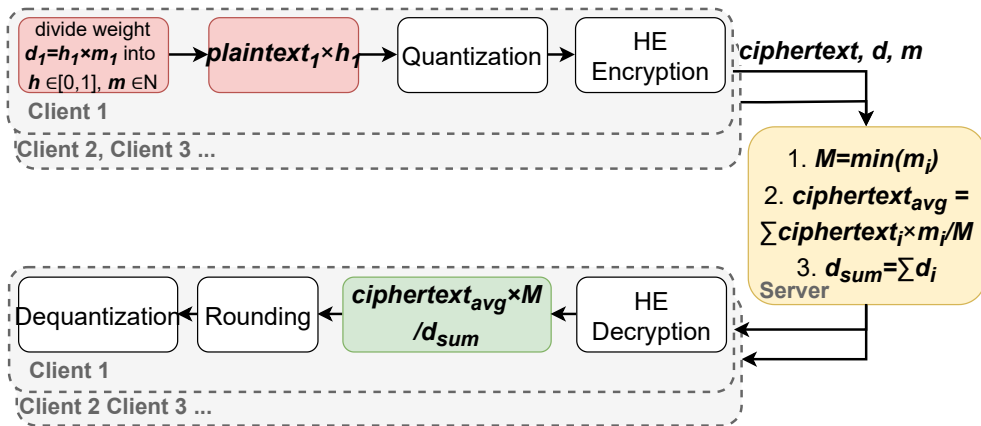


Figure 4.6: Combine MWAvg and HE encryption together.

By using *Flashev2* with *MWAvg*, this issue can be largely mitigated with nearly no extra overhead. As in the working flow shown in Fig. 4.6, the quantization is put after weighting, thus *Flashev2* can always use the same number of bits for quantization. Another advantage is provided by the design of  $m$ . As defined in Equation 4.10, all the weights that are located in the

range  $(2^{t-1}, 2^t]$  are transformed to the same  $m = 2^t$ . Therefore, as long as the weights of clients are in the same range, the masks are still able to cancel each other. However, the worst case is that each client uses a different  $m$ . In this case, the masks can not be canceled and will cause the same time complexity as the primary weighted averaging discussed before.

# Chapter 5

## Implementation

Following the methodology and design described in Chapter 3, we integrate the introduced four HE schemes, quantizations, FL training process, and secure protocols together and developed our framework named Homomorphic Encryption Federated Learning FLower Plugin (Heflp). Heflp is one Python module that achieves PPFL by HE based on Flower[4] FL framework. Currently Heflp enables four HE schemes, as discussed in Section 4.3. What is more, to make it more usable, Heflp supports both Pytorch\* and Tensorflow†, which are two prevalent ML frameworks.

This chapter provides technical details about the structure of Heflp and its implementation and usage as well. Section 5.1 introduces the submodules of Heflp respectively while Section 5.2 shows how to use Heflp. Section 5.3 describes the steps to deploy Heflp into clients and TEE server.

### 5.1 Heflp Submodules

Heflp consists of five submodules, each designed to fulfill specific functions.

#### 5.1.1 heflp.secureproto

This submodule includes all the dependencies of HE ciphers and quantizers. It consists of two submodules: *heflp.secureproto.homoencrypschemes* and *heflp.secureproto.quantization*.

---

\*<https://pytorch.org/>

†<https://github.com/tensorflow/tensorflow>

### 5.1.1.1 homoencrypschemes

*heflp.secureproto.homoencrypschemes* contains the ciphers of HE schemes. In *.cypher*, a *CypherBase* class is defined as a paradigm, which includes three basic methods:

1. *encrypt*: encrypting the received plaintext and returning the corresponding ciphertext.
2. *decrypt*: decrypt the ciphertext and return decrypted plaintext.
3. *get\_seed*: get the seed of this cipher and ciphers with the same seed have the same behavior.

All the other ciphers defined for specific HE scheme inherit from it. Currently, *HeFlp* has implemented four kinds of ciphers: *Flashe*, *Flashev2*, *CKKS*, and *BFV*.

As *Flashev2* is one extensional version of *Flashe* while *Flashe* can be considered as one special case of *Flashev2* as explained in Section 4.3.2, both *Flashev2* and *Flashe* have similar cipher design. For both *FlasheCypher* and *Flashev2* the encrypted message comprises two parts, one Numpy array and one *FlasheCypherParams* object which contains the metadata of the encrypted message. The encrypting and decrypting processes follow the principle of *Flashe*, computing masks and adding them to the plaintext or removing them for decryption. The seed represents the key of the pseudorandom function used for mask generating. However, we optimized our proposed *Flashev2* to accelerate mask-generating by leveraging the build-in *int16* and *int32* data types, i.e. the speed of generating mask is approximately 2x faster when the bit-width parameter of *Flashev2* is 16 or 32.

Cipher *CKKS* and *BFV* are realized by a HE tool called *Pyfhel* [63]. So *CKKSCypher* or *BFVCypher* is developed as a wrapper of *Pyfhel* ciphers. However, as *Pyfhel* ciphers have a limitation of the maximum size of plaintext for each round of encryption, the two *HeFlp* ciphers are designed to use *Pyfhel* cipher to encrypt plaintext of arbitrary size by splitting it to multiple pieces before encryption if the size exceeds the maximum limitation. To transmit the ciphertexts from clients to the server, it is required to transform the ciphertexts into bytes and transform them back on the server side. Another class *CKKSHelper* (or *BFVHelper*) is developed to handle this kind of transforming task. The *CKKSCypher* (or *BFVCypher*) is only used by clients but the *CKKSHelper* (or *BFVHelper*) is required by both clients and the server.

### 5.1.1.2 quantization

This submodule contains two Python files. *quantizer.py* defines the symmetric *Quantizer* class introduced in Section 4.4.1 while *mwavg.py* defines the *MWAvQuantizer* class that inherits the *Quantizer* but enhances its ability by MWAvg discussed in Section 4.4.3. Besides, as MWAvg requires additional information  $m$  and weight  $w$  for each client to be transmitted to the server, another data class *MWAvParams* is defined to store and transform this kind of information.

### 5.1.2 heflp.training

This submodule includes the classes and methods for assisting model training. As the model parameters need to be transformed into a 1D array before quantization and encryption, *heflp.training* has one method *flatten\_model\_params* to extract parameters from the ML model object and flatten them into one 1D array and the reverse method *unflatten\_model\_params* to apply the flattened parameters to the given ML model.

To separate model training from other steps, a specific class *Runner* is defined to handle model training. To adjust different frameworks (Pytorch or Tensorflow), different types of the instantiated *Runner* can be used. Heflp already implements *PytorchRunner* and *TensorflowRunner* to train and validate the corresponding types of ML models. Besides, one *FakeRunner* is provided as well for testing only. *FakeRunner* will not update the model parameters but just "pretend" to train the model. In this way, the model training can be skipped without changing much code when testing other functionalities of Heflp, e.g. testing the validity of a new HE scheme, saving time and making the framework more flexible and extendable.

### 5.1.3 heflp.client

Heflp provides several client classes for different HE schemes (*FlasheClient*, *Flashev2Client*, *BFVClient*, *CKKSClient*), each of which is instantiated with several arguments, e.g. cipher, quantizer, ML model, runner, and fit epochs of each round. To keep our realization of *Flashe* the same as the origin, the *FlasheClient* uses the original *Flashe* scheme and primary weighted averaging strategy (described in Section 4.4.2). Both *Flashev2Client* and *BFVClient* apply the proposed MWAvg strategy (described in Section 4.4.3).

The *heflp.client* classes inherit the Flower client class. They define how to train and evaluate the ML model for each round of FL, HE encryption and

decryption processes, and data transmitted to and from the server.

Besides, Heflp also provides a *BasicClient* that does not implement any HE scheme and conducts the FL process only. It is used as a baseline, making it easy to compare the speed and traffic overhead between it and the clients that have HE enabled.

Each Heflp client follows a similar structure but has its own characteristics, e.g. the transmitted metadata are various (see Table 5.1).

Table 5.1: Metadata transmitted from clients to server

Client Type	Item name	Description
Basic	None	
Flashe	flashe_params	Flashe params: $S$ and $i$
Flashev2	mwavg_params	MWAvg params: weight $w$ and factor $m$
CKKS	None	
BFV	mwavg_params	Same above

#### 5.1.4 heflp.strategy

Similarly to *heflp.client*, on the server side, there are multiple strategies (*FlasheStrategy*, *Flashev2Strategy*, *BFVStrategy*, *CKKSStrategy*, *BasicStrategy*), each of which matches a specific HE scheme. One strategy plays two roles:

1. **Aggregator:** define how to process the received model updates and aggregate them.
2. **Controller:** control the FL training process by storing the state of training and specify the configurations (see more details in Table 5.2) sent to each client according to it.

The *heflp.strategy* classes inherit the Flower strategy class and thus have many arguments like the minimum number of clients for training, the minimum number of clients for evaluation, the fraction of clients for each round of training, etc. By specifying these arguments, the administrator is able to control the FL training behavior manually.

#### 5.1.5 heflp.utils

This submodule includes the tools used for logging, performance evaluation, parsing, etc. Heflp uses two kinds of loggers. The first logger is named *heflp*

Table 5.2: Configurations transmitted from server to clients

Client Type	Item name	Description
Basic	1. epochs_per_round	1. Num of epochs run in this round
Flashe	1. epochs_per_round	1. Same above
	2. i	2. Client id
	3. j	3. Current round number
Flashev2	1. epochs_per_round	1. Same above
	2. flashev2_params	2. Params for mask generation: $S$
	3. mwavg_params	3. MWAvg params: sum of weights and $M$
CKKS	1. epochs_per_round	1. Same above
	2. mwavg_params	2. Same above
	3. ckks_context	3. Context for CKKS cipher initialization
BFV	1. epochs_per_round	1. Same above
	2. mwavg_params	2. Same above
	3. bfv_context	3. Context for BFV cipher initialization

and used to record general messages, like the beginning and end of each phase, choices of clients for each round of FL training, etc. The other logger named *heflp-eval* mainly focuses on logging the information for evaluation, such as the time cost of encryption and decryption.

*heflp.utils* also has tools to make evaluation simpler. For example, we defined one *TimeMarker* class for marking the timestamps of actions and calculating the intervals automatically, making time cost evaluation much easier.

### 5.1.6 heflp.info

Besides the submodules introduced above, Heflp also has one *info.py* file that contains configuration information of Heflp such as the names of loggers, log format and level, supported schemes, and ML frameworks.

## 5.2 Heflp Usage

It is very convenient to use Heflp building the PPFL training on multiple clients and the TEE server by several lines of code. As shown in Fig. 5.1, after initializing the server and multiple clients, they will communicate with each other and the FL training will start automatically.

On the server side, the only thing it needs to do is to initialize a Heflp

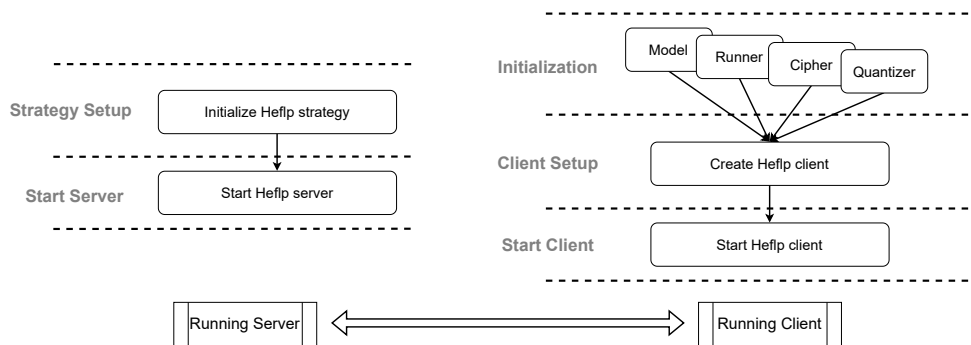


Figure 5.1: Usage of Heflp

strategy with customized parameters and pass it to Heflp method *start\_server* to run the server inside the TEE.

For each client, as discussed in Section 5.1, firstly the users need to initialize some necessary components. Then pass these components into one Heflp client and initialize it. Finally, start the client and specify the server address by using the *start\_client* method provided by Heflp. Then the client will communicate with the server automatically as long as the server address is correctly set.

Considering that there are already paradigms of cipher, runner, client, and strategy, Heflp is easy to be expanded with more features. We wrote README of guidelines, making it friendly for new users to start as well.

### 5.3 Deployment

To deploy Heflp, the device should have python ( $\geq 3.8$ ) and pip installed. For the client, the only thing it needs to do is to pull the repository and install the necessary dependencies. If it is done correctly, Heflp itself should be installed as a local module as well.

For the server, considering that TEE has a very limited memory resource, it is necessary to make the installed framework as light as possible to save memory. To minimize the memory footprint, we wrote one bash script *extract-server.sh* to extract only the necessary submodules and pack them into one package named *heflp-server*. Because the server does not train the ML model but aggregates the model parameters only, all the dependencies like Pytorch and Tensorflow are not needed and will not be installed on the server. By avoiding installing these redundant dependencies, we can save memory of approximately 1.6G. Therefore, to deploy the Heflp to the TEE server, the

administrator first runs the bash script on any machine with bash enabled, then uploads the extracted *heflp-server* to the server. Finally, install all the necessary dependencies.

In this project, we already developed one program *fl-server* with default settings. The administrator can directly run it inside TEE. It is also very convenient for users to develop their own program following this example.



# Chapter 6

## Results and Evaluation

This chapter presents the simulation results and conducts the evaluation of the framework Heflp on them. The evaluation is divided into three steps:

1. **Security Assessment:** Assess if the framework is able to defend the adversaries defined in Section 3.1.
2. **Performance Evaluation:** Evaluate the model performance of FL with and without HE, aiming to validate that the additional secure mechanisms do not sacrifice the benefits of model training.
3. **Efficiency Evaluation:** Evaluate the efficiency loss of Heflp using different HE schemes and the effects of TEE as well. By considering various scenarios, this kind of experiment can help choose the most appropriate setting in light of specific demands.

The security evaluation is discussed in Section 6.1. Section 6.2 presents the training results and conducts the performance evaluation. Section 6.3 shows the detailed efficiency evaluation.

### 6.1 Security Assessment

As defined in Section 3.1, there are two adversaries this project considers. The server adversary has root access to the cloud platform and aims to violate the code integrity and data confidentiality. The client adversary can eavesdrop on the communication between the server and other clients. The goal of this adversary is to obtain the sensitive model parameters of other benign clients.

To defend the server adversary, Heflp mainly relies on the combination of TEE and HE. Because the aggregation code and the sensitive client model

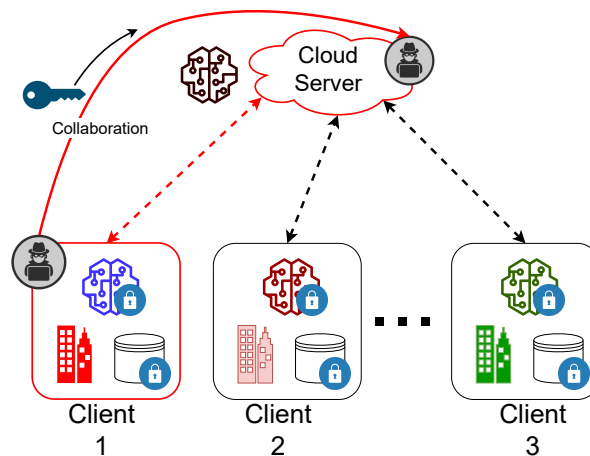


Figure 6.1: collusion of the server and client adversaries.

updates are stored and processed in the TEE, whose memory and process are isolated from outside, even the cloud platform itself can not tamper with the code running inside and the model updates if it does not have the permission. By leveraging this property, the integrity of code and data is under protection. On the other hand, clients use HE cipher to encrypt the model parameters before sending them to the aggregation server, making the data secure during the whole process of transmission and aggregation. Therefore, HE guarantees the data confidentiality. By leveraging TEE and HE together, we thus ensure both integrity and confidentiality of code and data.

Considering the client adversary, Heflp defends the malicious behavior by using TLS as well as HE. Any attackers that do not have the key of HE can not obtain the sensitive data even if they can eavesdrop on the communication as the model parameters are encrypted by HE. However, considering a more dangerous adversary that pretends to be a valid client participating in FL like all the other authenticated and trustworthy clients, HE is not enough because currently Heflp requires all the clients to use the same key for HE encryption and decryption. Therefore, if the adversary client obtains the message of another benign client, it is able to decrypt and get the model parameters of that victim client using the same HE key. To defend against this kind of malicious behavior, Heflp implements TLS to enhance the protection of data confidentiality and integrity during the transmission. TLS is used to establish a secure communication channel between the server and each client. When they communicate with each other, the messages are encrypted and a digital digest (MAC) calculated from the message is attached as a fingerprint to guarantee the data integrity.

However, the collusion of these two kinds of adversaries is not tackled by Heflp and can be marked as a limitation of Heflp. If one malicious user that obtains the private key of HE shares the key to the server (as illustrated in Fig. 6.1), the server is able to obtain the encrypted data and decrypt it by the key, violating the data confidentiality. In the current stage, Heflp does not consider the situation that the server adversary knows the private key used for HE. Although defending this kind of attack is out of the scope of this project, we set it as part of our future work.

## 6.2 Performance Evaluation

HE will cause a loss of result accuracy when conducting operations on the ciphertexts. The reason is that CKKS can only obtain an approximate result while other integer HE schemes (Flashe or BFV) require quantization, resulting in a loss when dequantizing the result. Although ML model parameters are not sensitive and can tolerate a certain level of loss, it is still necessary to verify how much this kind of loss will impact the model performance. In this section, we first measure the error rates of HE schemes and then get the training curve under different HE settings by simulating the whole FL process.

### 6.2.1 Error Rates of HE Schemes

To evaluate the effects of HE schemes on the model training, one experiment was conducted to measure the error rate of aggregation results caused by each of the HE schemes that Heflp supports. The experiment follows the principle that ideally the aggregation results  $A'_N$  of PPFL applying HE for data security should be exactly the same as the results  $A_N$  of FL without HE protection under the same setting ( $N$  denotes the total number of model parameters). Therefore, the errors  $E_N$  can be represented as the difference between the  $A'_N$  and  $A_N$ , denoted by  $||A'_N - A_N||$ . The collected  $E_N$  is used to calculate the error probability distribution while the error rate is simply defined as the mean value of  $E_N$ . Considering that the weights of clients will affect the performance of MWAvg, the experiment is divided into two tests, a uniform test and a nonuniform test. Because the integer HE schemes (Flashe, Flashev2, BFV) themselves provide exact accurate results as long as they are set properly, for these three schemes, quantization becomes the main reason for the error. Therefore in the two tests, three r-bit quantizations are evaluated

( $r = 16/18/20$ ). Theoretically, the more bits used for quantization, the less the error.

Name	Value	Description
Model Type	CNN	The model type used for FL
Model Scale $N$	50890	The number of model parameters
Client Number	5	Number of clients participating
Client Weights	500, 500, 500, 501, 502	The weight of each client respectively

Table 6.1: The settings of the uniform test

The uniform test sets all the clients equally, thus all clients share approximately the same weight 500 (as an example, actually any weight value is valid). In this case, the aggregation can be considered equal to simply calculating the average result of all received model updates. The detailed settings of this test are listed in Table. 6.1. The results are shown in Fig. 6.2a. From the results it is easy to observe that in this uniform setting the three integer schemes (Flashe, Flashev2, BFV) have almost the same error rate (the orange line marks) and probability distribution, which is because the multiplicative factor before quantization is  $\frac{d}{m} = \frac{500}{2^9} = 0.977 \rightarrow 1$  (See more details in Section 4.4.3). When  $r = 16$  the error rate of CKKS is 5 times smaller than the other three. However, the error rates of integer schemes significantly reduce with the increase of  $r$ . The principle is that when  $r$  increases  $b = 2$ , the error rate reduces to its  $\frac{1}{2^b}$ , conforming to the quantization principles we have discussed in Section 4.4.

Name	Value	Description
Model Type	CNN	The model type used for FL
Model Scale $N$	50890	The number of model parameters
Client Number	5	Number of clients participating
Client Weights	1000, 2000, 3000, 4000, 5000	The weight of each client respectively

Table 6.2: The settings of the nonuniform test

The nonuniform test allocates various weights to the clients, i.e. client 1 has the smallest weight of 1000 while client 5 has the largest weight of 5000, thus the weighted averaging strategy plays an important role in the aggregation under this setting. The detailed settings are listed in Table. 6.2 and the results are shown in Fig. 6.2b. Compared to the uniform test, in nonuniform settings,

the error of Flashev2 and BFV increases and is a bit larger than Flashe. It is brought by the MWAvg strategy as for some weights like 3000 and 5000 the pre-quantization multiplicative factor  $\frac{d}{m}$  is only 0.7 or smaller, causing an increase of error. As explained in Section 4.4.3 the maximum error rate of MWAvg is twice that of the primary weighted averaging, according to the test results. Another observation is that when the scheme is Flashe while  $r = 20$  the error rate is unexpectedly higher than all the others. This outlier indicates that the aggregation is overflowing. When  $r = 20$  and the sum of weights is  $1000 + 2000 + 3000 + 4000 + 5000 = 15000 = 2^{13.87}$ , so the theoretically maximum value of the sum results is  $2^{(20-1)} \cdot 2^{13.87} = 2^{32.87} > 31$  using primary weighted averaging strategy. As we use 32-bit signed integers to cache the aggregation results, more than half of the values are overflowing when using Flashe. However, both Flashev2 and BFV that apply MWAvg and CKKS can mitigate this overflowing problem, conforming to the principles of MWAvg and its goals explained in Section 4.4.

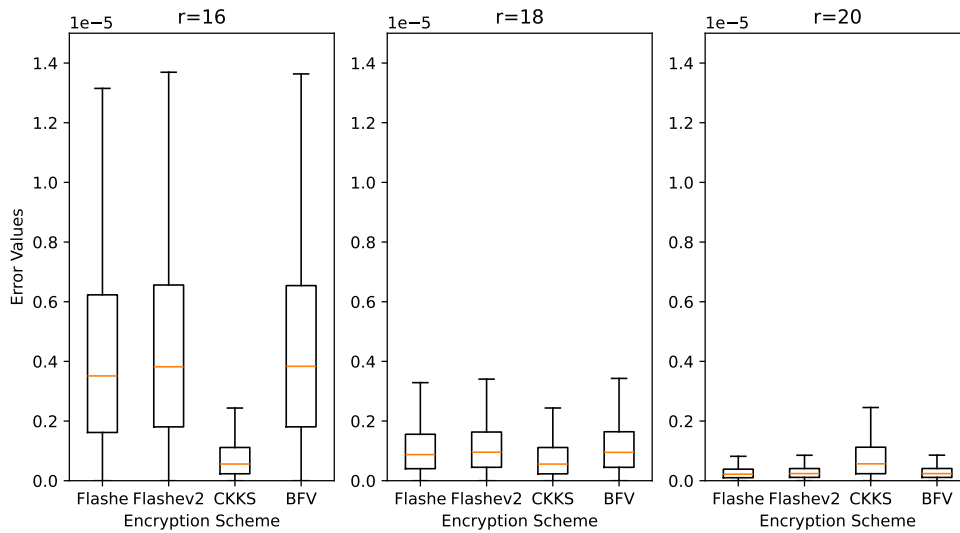
## 6.2.2 Training Performance Evaluation

### 6.2.2.1 Simulation settings

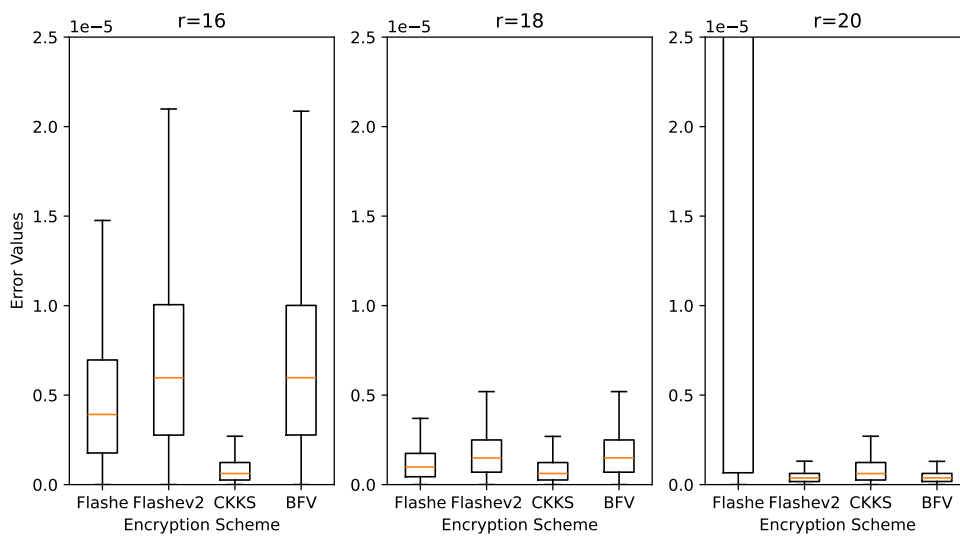
To evaluate the effects of HE mechanisms on the FL training process, two simulations were run on the MNIST dataset and the TI dataset (see more in Section 3.3.3). One VM instance with AMD SEV-SNP supported is set as the FL server while another VM instance simulates three clients. The simulated clients connect to the server and they set up the FL training process together. To monitor the training curve, the performance of the global model is recorded after each round of training. For the test on the MNIST dataset, the model used for training is a 5-layer Convolutional Neural Network (CNN) model and the evaluation metric is the accuracy (or *Acc*) defined as the number of correct classifications divided by the total number of test samples. The test on TI dataset uses a Long Short-Term Memory (LSTM) for training and chooses Mean Squared Error (MSE) as the evaluation metric. What is more, the centralized training without FL is also run on the client VM in order to compare the performance between the FL and centralized training. The detailed simulation settings are shown in Table 6.3.

### 6.2.2.2 Training Curves and Analysis

After running the simulation, the results of the two tests are shown in Fig. 6.3. Each kind of FL training is denoted by one mark while the blue dash line

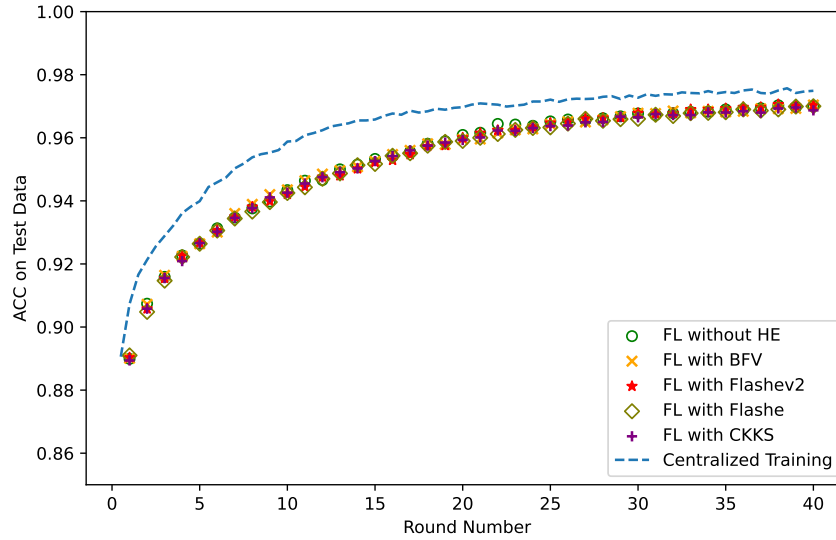


(a) Uniform test

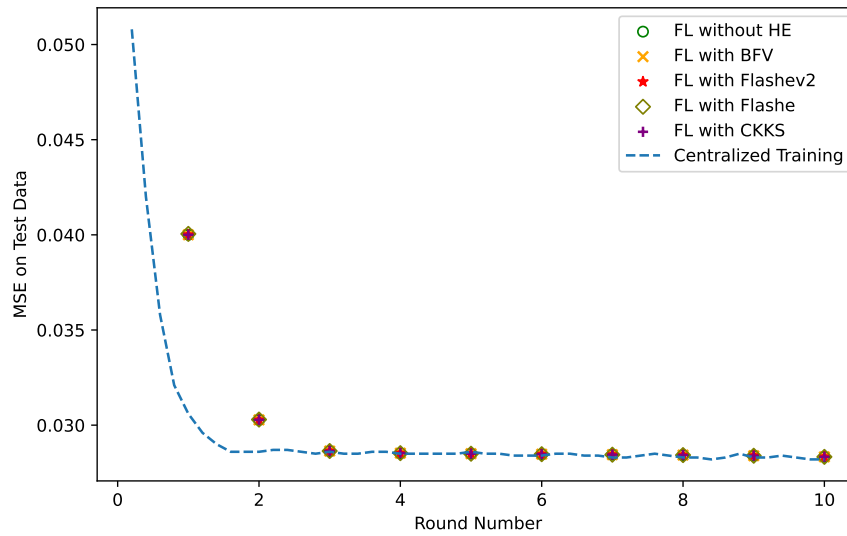


(b) Nonuniform test

Figure 6.2: (a) The results of the uniform test, where the weights of all clients are approximately the same. (b) The results of the nonuniform test, where the weights of clients vary.



(a) Training curve on MNIST dataset



(b) Training curve on TI dataset

Figure 6.3: (a) The training curve of the CNN model using metric Acc on MNIST dataset. (b) The training curve of the LSTM model using metric MSE on TI dataset.

Name	Value	Description
Model Type	MNIST: CNN TI: LSTM	The model type used for FL
Available Clients	3	Number of available clients
Training Clients	2	Number of active clients per round
Quantization Bit	16	r-bit for quantization
Training Rounds	MNIST: 40 TI: 10	Number of FL training rounds
Epochs per Round	MNIST: 2 TI: 5	How many epochs are trained per round
Learning rate	MNIST: 0.01 TI: 0.001	The learning rate for model training
Metrics	MNIST: Acc TI: MSE	The metrics used for evaluation

Table 6.3: Settings for the simulation of FL on MNIST or TI dataset.

shows the curve of centralized training. The test results of FL are scattered because the global model is updated only after a round of training, which comprises more than one training epoch, and its performance is evaluated after the aggregation of every round. However, for centralized training, the model is evaluated after each training epoch, making the evaluation more consistent. From the curves, it is clear that for both FL training on the MNIST dataset and TI dataset, the model has a similar evaluation result no matter if it applies HE and which HE scheme is used. This observation confirms that under the settings given by Table 6.3 the HE security mechanisms do not sacrifice the benefits of FL training, and the model can still retain its performance with the protection of HE. Besides, we can find that the FL training has a slower convergence speed compared to the centralized training. There are several reasons. For instance, the aggregation algorithm we used is the classic *FedAvg* and the convergence speed can be increased by using more advanced algorithms (like *FedProx*) for specific use cases. On the other hand, the training parameters like optimizer, learning rate and epochs per round have not been carefully adjusted. The task itself also affects the model convergence, i.e. the FL training on the TI dataset converges soon while the FL training on the MNIST dataset takes many more rounds. However, in this work, we do not concentrate on improving the performance of FL but on validating the practicality of the security mechanism based on HE and TEE. We believe more advanced FL methods can boost the model performance and it can be marked

as one future work to combine our framework with other FL algorithms.

## 6.3 Efficiency Evaluation

Besides the ML training performance, it is also critical to evaluate the efficiency of the supported HE schemes in various settings since different HE security settings have their own characteristics and perform differently. This section discusses the evaluation results and corresponding analysis first. Then a comparison of the pros and cons of the HE schemes is given so that users can have some evidence to trade off and select proper security settings according to their specific demands.

### 6.3.1 Evaluation Settings and Metrics

Because of the constraint of VM resources, it is impossible for us to simulate many clients with full functionalities, i.e. if simulating 10 clients for the whole FL training, there will not be enough running memory and computational resources to run all the client processes in parallel. In the setting of this evaluation, the clients do not train the models locally but do the encryption and decryption only. As this evaluation does not care about the model performance of FL while the validity of HE protection has been proved in Section 6.2, it is reasonable to skip the local training part to save time and resources. And other operations (e.g. quantization, encryption, aggregation, and so on) are unchanged. The FL training will not impact their efficiency as well, as long as the quantization and HE encryption parameters are configured to avoid the overflowing problem.

To evaluate the efficiency of different HE schemes, the average time costs of encryption and decryption of one client and the aggregation are recorded respectively. The sizes of the ciphertexts are collected as well. The ciphertext size can not only indicate the memory overhead but also represent the traffic overhead because the ciphertext is the major content of transmitted messages. By collecting information of the time overhead and memory overhead, it is able to measure the time and spatial efficiency under various settings reasonably.

When conducting the experiments for various scenarios, three main aspects of use cases are considered:

1. Model Type: three types of ML models of different scales are selected for experiments. They are a 5-layer CNN (50890 parameters), A LSTM model (220355 parameters), and a ResNet18 model (11689512

parameters). The scale of the model impacts both the time overhead of encryption and aggregation and the traffic overhead because a larger model always results in a larger ciphertext and requires more time for encryption. What is more, the aggregation will also comprise more homomorphic operations, thus its time cost is higher as well.

2. Number of clients: As this project only considers a cross-silo scenario where the number of clients is limited (See more in Section 3.1), we conduct our experiments when client number is 2 / 5 / 10 / 20. The total number of clients can impact the efficiency of aggregation as the more clients participating per round, the more model updates need to be aggregated. However, it does not affect the efficiency of HE encryption and decryption and the traffic overhead as long as using the same HE scheme and model.
3. Weights of clients: The weights can influence MWAvg and therefore have a relatively big impact on the performance of the Flashev2 scheme (As explained in Section 4.4.4). In the experiments, two kinds of weight patterns are considered as what we set before in the performance evaluation: the uniform setting where all the clients share the same weight (1000) and the nonuniform setting where the weights of different clients vary (in [1000, 5000]).

### 6.3.2 Time Efficiency Test of Uniform pattern

This test aims to evaluate the effects of the number of clients and their weights, thus we fixed the model type as LSTM. Two batches of test units are done according to the uniform or nonuniform weight patterns. In the uniform pattern, all the clients are set to the same weight 1000, thus the aggregation is to simply calculate the average of all the model updates. The results in Fig. 6.4 show that the time overheads of HE schemes differ very much. Comparing the total time costs, the Flashe has the highest time cost when the clients are few but CKKS becomes the most time-consuming choice when there are more than 10 clients. The total time cost of Flashe and Flashev2 almost remains the same no matter the number of clients but for CKKS and BFV the more clients the higher the total time cost. By analyzing the composition of the time cost of each HE scheme, we can find that Flashe and Flashev2 almost spend no time to aggregation while the time cost of aggregation of CKKS and BFV increases with the increasing of clients, causing higher total time cost. As explained in Section 4.3, CKKS and BFV have a shortcoming in that they take much time

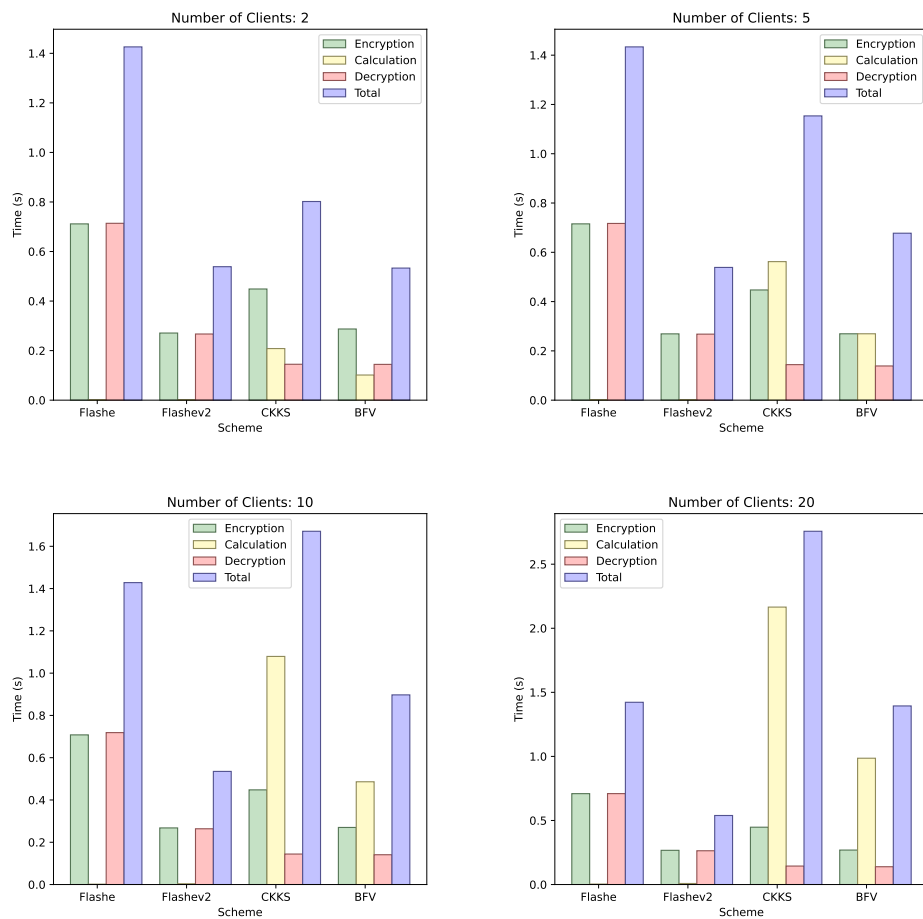


Figure 6.4: Results of the time efficiency test when all the clients have the same weight 1000. The time costs of encryption, decryption, and aggregation (calculation) are recorded while the total time cost is the sum of three, representing the total time cost of one client.

to execute homomorphic operations when aggregating model updates, which is the reason for the increase in time cost. However, Flashe and Flashev2 only sum up the received model updates without any additional computational cost, thus significantly saving the time of aggregation. Although they consume a relatively longer time for encryption and decryption, the advantage becomes obvious when the number of clients is large, especially for Flashev2.

The Flashev2 is developed based on the original version of Flashe and we optimized the code to accelerate the mask-generating when the bit width of Flashev2 is set to 16 or 32. As explained in 4.4.4, Flashe requires more bits for encryption but Flashev2 does not. Therefore, the mask-generating speed of Flashev2 is much higher than Flashe. According to the cryptosystem of Flashev2, when the weights of all clients are the same, most of the generated masks can be canceled by each other, making decryption efficient. We can also observe that BFV takes less time than CKKS. However, in this project, we did not dive much into the principles of these two schemes. One reason could be that BFV is an integer HE scheme while CKKS needs more complicated operations to calculate the weighted average results of the encrypted real-number model updates.

### 6.3.3 Time Efficiency Test of Nonuniform Pattern

In the nonuniform pattern, the weight of each client is selected from the range [1000, 5000], which influences the weighted averaging during aggregation. From the results shown in Fig. 6.5 and compare it with the uniform pattern, we can get the same observation that Flashe and Flashev2 still have the property to avoid the increase of time cost when there are more clients, but BFV and CKKS do not have such an advantage. In the nonuniform pattern, BFV is still faster than CKKS in all cases which is mainly because BFV spends much less time on aggregation than CKKS. What is more, Flashe can retain the same efficiency when weights become nonuniform but Flashev2 is much slower than the case where weights are uniform. From the figure, it is clear that the encryption and decryption of Flashe have approximately the same time cost no matter if the client weights are uniform, which is because the client weights do not impact the Flashe mask-canceling mechanism. But for Flashev2 the mask-canceling can be impacted when the weights are significantly different, i.e. if for client  $i$  and  $i + 1$ , the MWAvg multiplicative factor  $m_i \neq m_{i+1}$ , then the Flashev2 masks can not be canceled by each other, causing a higher time cost of decryption than encryption (See more details in Section 4.4.4). However, even in this nonuniform pattern, Flashev2 is still able to gain a lower time cost

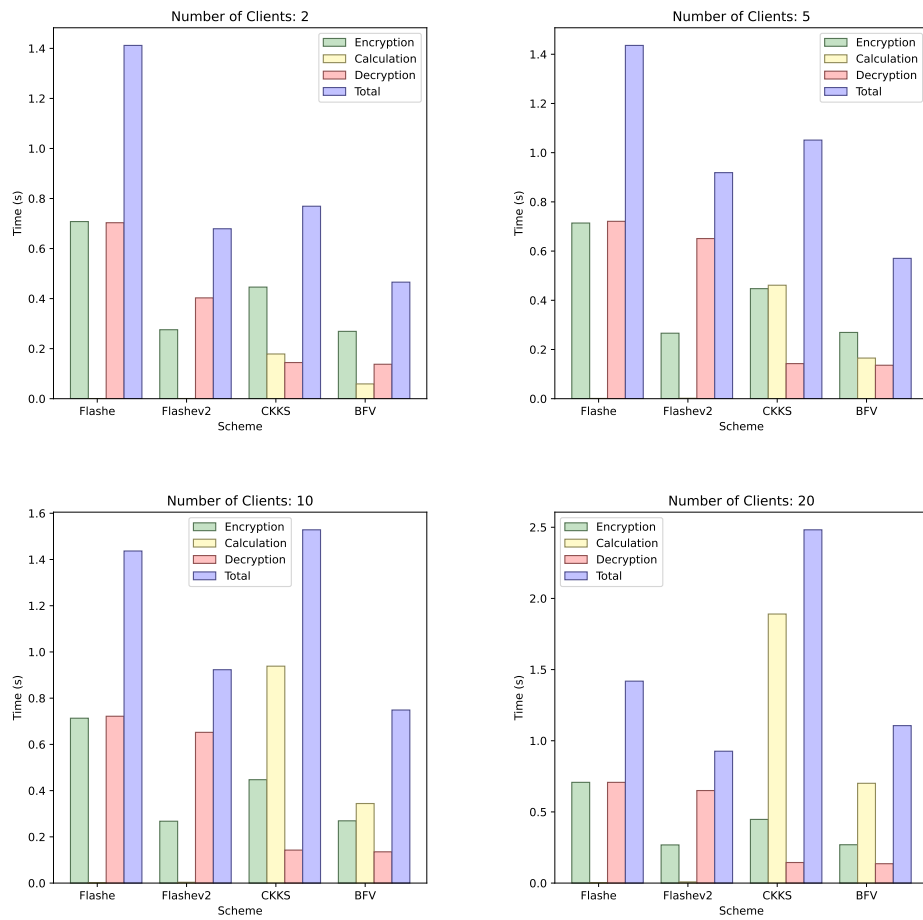


Figure 6.5: Results of the time efficiency test when the weights of clients are in range  $[1000, 5000]$ .

than Flashe, making Flashev2 more competitive.

### 6.3.4 Time Efficiency Test of Model Types

To evaluate the effects of model scales on the time costs, three types of models are tested and we set the number of clients to 5 and the nonuniform pattern of weights. The evaluation results are shown in Fig. 6.6.

The models we tested are a 5-layer CNN model, an LSTM model, and a ResNet18 model. Only considering the parameter numbers, the ResNet18 model is about 53 times larger than the LSTM model, while the LSTM model is about 4 times larger than the CNN model. From the results, the total time costs of each HE scheme also have approximately the same proportions. Therefore, we argue that there is a linear relationship between the time cost and the number of model parameters, which applies to all HE schemes.

From Fig. 6.6 we can also observe that the three histograms look very similar. Although the time cost significantly differs when the tested model is changed, the relative values of time costs between HE schemes have the same form, which confirms that the time cost of HE schemes follows the same principle no matter which model is used.

### 6.3.5 Memory Overhead Test

To evaluate the memory overheads of the 4 HE schemes, we collected the byte data of models when transmitting to the server and compared them to the model without HE. The evaluation of memory overheads of the 4 HE schemes and the comparison to the model without HE are shown in Fig. 6.7. From the three histograms of models of different scales, we can observe that Flashe and Flashev2 can significantly save memory resources compared to CKKS and BFV. As introduced in Section 4.3, Flashe and Flashev2 add masks directly to the plaintext and thus do not increase the size of ciphertext, but the encryption of BFV and CKKS relies on complex cryptography primitives to support homomorphic operations, making a ciphertext about 32 times larger than the plaintext. Therefore, we argue that Flashe and Flashev2 are over 30 times more spatial-efficient than BFV and CKKS. What is more, this test also shows that BFV and CKKS are not good choices when there are many active clients and the model scale is big, as the cloud server may not have enough memory to cache all the received encrypted model updates, raising the risk of program crashes.

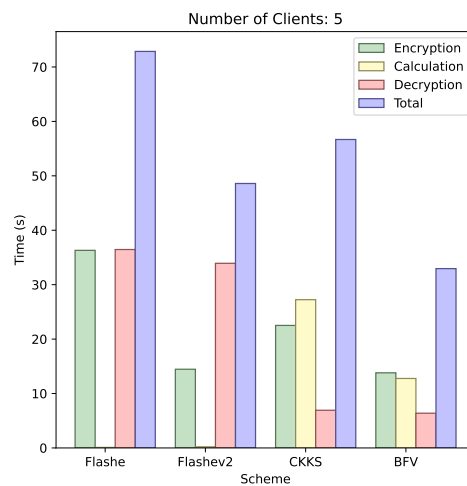
Note that the overhead of Flashev2 is slightly different from Flashe and the



(a) CNN



(b) LSTM



(c) ResNet18

Figure 6.6: Results of the time efficiency test for three models: CNN (50890 parameters), LSTM (220355 parameters), ResNet18 (11689512 parameters).

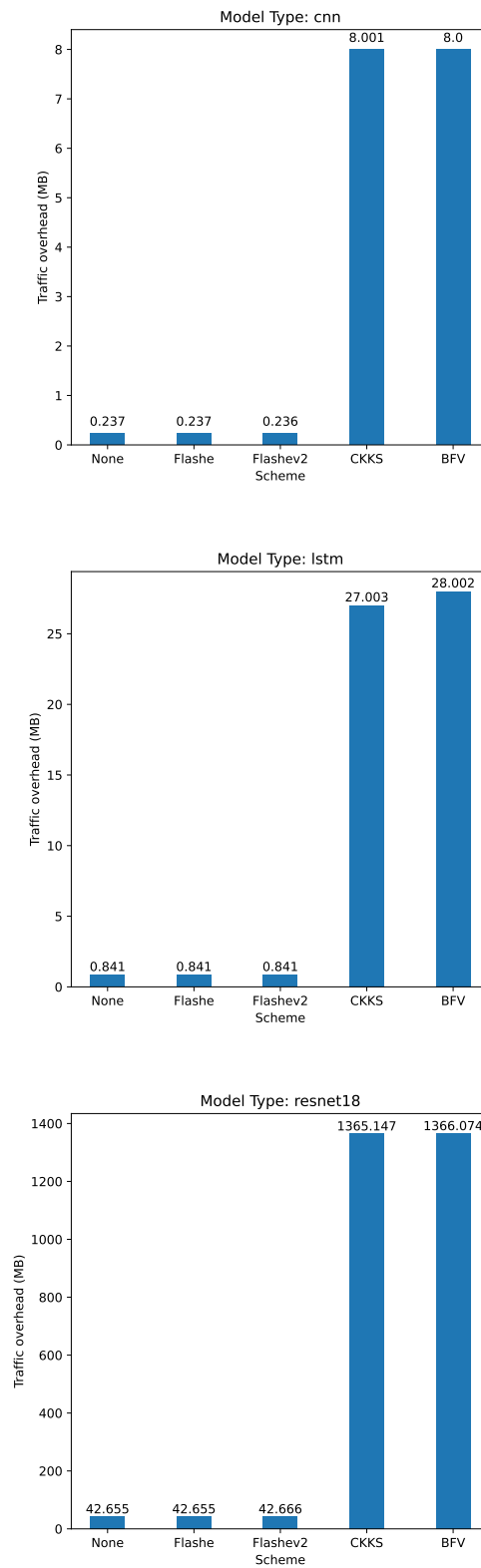


Figure 6.7: Results of the memory overhead test for three models: CNN (50890 parameters), LSTM (220355 parameters), ResNet18 (11689512 parameters). *None* means the original size of the model without using any HE scheme

original model, which is because when developing Flashev2 we used another encoding tool to transform the model into bytes.

### 6.3.6 HE Efficiency Comparison

After conducting the tests for both time and spatial efficiencies, we summarized the pros and cons of all the four HE schemes in the context of FL training:

1. **Flashe:** According to our experimental results, Flashe does not have obvious advantages compared to the other three schemes. But for some special cases where both the weights and the number of clients are small, e.g. weights are integers and smaller than  $2^{10} = 1024$  while clients are fewer than  $2^6 = 64$ , Flashe achieves a stable performance as it is not weight-sensitive while retaining a high level of spatial efficiency. However, it is not fast and therefore not a good choice for latency-sensitive applications, and we mark it as our future work to accelerate it by optimizing the code or parallel computing.
2. **Flashev2:** It has a good time and spatial efficiency especially when the client number is big. What is more, it can work well in wide scenarios as it supports arbitrary weights and more than one hundred clients, which is enough to cover all the cross-silo FL scenarios that we considered in this project. However, Flashev2 might suffer from a longer decryption time if the weights of clients vary in a wide range.
3. **CKKS:** As CKKS has neither the time efficiency nor spatial efficiency advantage, it should not be chosen unless there is a special reason, i.e. requires the real-number calculation without quantization or complex multiplicative operations, which might be needed in the future when we apply other FL aggregation algorithms to Heflp.
4. **BFV:** Based on the time efficiency tests, BFV has the advantage of short execution time when clients are few ( $<10$  in our experiments), and this advantage becomes obvious when the model scale is big. And similar to Flashev2, the client weights do not influence the performance of BFV as well. However, the high memory overhead and the significant increase in time cost with the increasing of clients weaken its application.



# Chapter 7

## Conclusions and Future Work

In this project, we design and develop one PPFL framework powered by HE and TEE named Heflp, and conduct a series of evaluations. We not only implement three existing HE schemes (Flashe, CKKS, and BFV) but also develop our own Flashe version 2 (Flashev2), achieving a higher flexibility, robustness, and execution speed with the combination of an improved weighted averaging strategy (MWAvg). The practicality of Heflp is proved by simulating the FL training process on the TI dataset and MNIST dataset and evaluating its security and performances correspondingly, which shows that Heflp is able to protect both data confidentiality and integrity during the whole lifecycle of FL training from the predefined adversaries while retaining the time and spatial efficiency if appropriate HE schemes are chosen.

### 7.1 Discussion

In this project, we have a strict assumption of the threat model, assuming adversaries can happen on both the server and client side, which leads to our solution Heflp to implement TEE and HE together to secure FL process. If using a looser threat model, i.e. considering only one kind of adversary, it is possible to apply simpler solutions. For instance, if the TEE is trustworthy, which means that it can provide full protection of data confidentiality and integrity, and all the clients trust the CSP providing FL aggregation service inside the TEE, HE is not necessary. It might be achieved in the future, requiring the TEE hardware manufacturers to improve their products and new designs to a higher security level. If all the clients' validity can be verified, some other strategies can be feasible as well. One possible solution to retain data integrity without TEE is to let a group of clients act as aggregators

and compare their aggregated results. If all the aggregated global models are the same, it indicates that the aggregation is correctly done. However, this approach requires clients to be honest while suffering from much higher time and computational overhead. In one word, when considering looser threat models, it is possible to use TEE or HE only for privacy protection. Nevertheless, discussing details of these possible alternatives is out of the scope of this project.

Besides, because of the limitation of time and resources, we did not conduct a deep work on TEE itself, i.e. we integrate TEE into our framework to provide stronger security guarantee while focusing on its application and deployment only, without discussing its hardware principle and technical details. We set it as one future work to explore more about various TEE products and their characteristics, which can potentially help to improve Heflp.

## 7.2 Future Work

In the future, some new features and optimizations are expected to be considered, aiming to mitigate the existing security vulnerabilities of Heflp explained in Section 6.1, accelerating the execution speed and expanding its functionalities.

Right now Heflp is still vulnerable to the complex attacks caused by the sharing key used for HE schemes. We plan to develop another version of Flashe (Flashev2) to support each client using its unique key for encryption, providing a higher level of protection against the risk of leaking keys.

What is more, Flashe and Flashev2 are realized by Python, which is not an efficient programming language, making them not run as fast as we expect. Therefore, one future work can be to rewrite (part of) the code in C language and pack them as one module. In this way, the running speed is expected to be accelerated by several times.

Another potential feature is to develop an automatic script to detect and select the proper HE scheme before starting the FL training to achieve the highest efficiency while saving memory resources.

To expand the ability of Heflp and adjust it to fit more FL scenarios, we expect to develop and implement more FL algorithms (like FedProx) and HE schemes to it. In addition, developing templates to leverage Heflp build-in classes and tools can also be promising and helpful since future programmers can develop their own features with the templates for their own needs.

## References

- [1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. PMLR, Apr. 2017, pp. 1273–1282, iSSN: 2640-3498. [Online]. Available: <https://proceedings.mlr.press/v54/mcmahan17a.html> [Pages 1, 14, 15, 16, and 41.]
- [2] D. Goltzsche, M. Nieke, T. Knauth, and R. Kapitza, “AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting,” in *Proceedings of the 20th International Middleware Conference*. Davis CA USA: ACM, Dec. 2019. doi: 10.1145/3361525.3361541. ISBN 978-1-4503-7009-7 pp. 123–135. [Online]. Available: <https://dl.acm.org/doi/10.1145/3361525.3361541> [Page 1.]
- [3] M. Sabt, M. Achemlal, and A. Bouabdallah, “Trusted Execution Environment: What It is, and What It is Not,” in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, Aug. 2015. doi: 10.1109/Trustcom.2015.357 pp. 57–64. [Page 1.]
- [4] A. Muñoz, R. Ríos, R. Román, and J. López, “A survey on the (in)security of trusted execution environments,” *Computers & Security*, vol. 129, p. 103180, Jun. 2023. doi: 10.1016/j.cose.2023.103180. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0167404823000901> [Page 2.]
- [5] C. Fontaine and F. Galand, “A Survey of Homomorphic Encryption for Nonspecialists,” *EURASIP Journal on Information Security*, vol. 2007, pp. 1–10, 2007. doi: 10.1155/2007/13801. [Online]. Available: <http://jis.eurasipjournals.com/content/2007/1/013801> [Pages 2 and 11.]

- [6] Z. Jiang, W. Wang, and Y. Liu, “FLASHE: Additively Symmetric Homomorphic Encryption for Cross-Silo Federated Learning,” Sep. 2021, arXiv:2109.00675 [cs]. [Online]. Available: <http://arxiv.org/abs/2109.00675> [Pages 2, 13, 18, 25, 29, and 41.]
- [7] “FederatedAI/FATE,” Apr. 2023, original-date: 2019-01-24T10:32:43Z. [Online]. Available: [https://github.com/FederatedAI/FATE/blob/9432615a9a8957dc2afda3363e32c8b9dfd3c963/doc/tutorial/ipcl\\_tutorial.md](https://github.com/FederatedAI/FATE/blob/9432615a9a8957dc2afda3363e32c8b9dfd3c963/doc/tutorial/ipcl_tutorial.md) [Pages 5 and 18.]
- [8] “Flower: A Friendly Federated Learning Framework,” Mar. 2023, original-date: 2020-02-17T11:51:29Z. [Online]. Available: <https://github.com/adap/flower> [Pages 5, 18, and 37.]
- [9] G. A. Reina, A. Gruzdev, P. Foley, O. Perepelkina, M. Sharma, I. Davidyuk, I. Trushkin, M. Radionov, A. Mokrov, D. Agapov, J. Martin, B. Edwards, M. J. Sheller, S. Pati, P. N. Moorthy, S.-h. Wang, P. Shah, and S. Bakas, “OpenFL: An open-source framework for Federated Learning,” *Physics in Medicine & Biology*, vol. 67, no. 21, p. 214001, Nov. 2022. doi: 10.1088/1361-6560/ac97d9 ArXiv:2105.06413 [cs]. [Online]. Available: <http://arxiv.org/abs/2105.06413> [Page 5.]
- [10] “TensorFlow Federated.” [Online]. Available: <https://www.tensorflow.org/federated> [Pages 5 and 18.]
- [11] C. Zhang, “batchcrypt,” Mar. 2023, original-date: 2020-05-27T10:01:04Z. [Online]. Available: <https://github.com/marcosz/BatchCrypt> [Pages 5, 19, and 33.]
- [12] “White Papers & Reports – Confidential Computing Consortium.” [Online]. Available: <https://confidentialcomputing.io/resources/white-papers-reports/> [Page 9.]
- [13] A. Viand, C. Knabenhans, and A. Hithnawi, “Verifiable Fully Homomorphic Encryption,” Feb. 2023, arXiv:2301.07041 [cs]. [Online]. Available: <http://arxiv.org/abs/2301.07041> [Pages 9 and 13.]
- [14] M. Pei, H. Tschofenig, D. Thaler, and D. Wheeler, “Trusted Execution Environment Provisioning (TEEP) Architecture,” Internet Engineering Task Force, Internet Draft draft-ietf-teep-architecture-19, Oct. 2022, num Pages: 38. [Online]. Available: <https://datatracker.ietf.org/doc/draft-ietf-teep-architecture> [Page 9.]

- [15] S. Pereira, D. Cerdeira, C. Rodrigues, and S. Pinto, “Towards a Trusted Execution Environment via Reconfigurable FPGA,” Jul. 2021, arXiv:2107.03781 [cs]. [Online]. Available: <http://arxiv.org/abs/2107.03781> [Page 9.]
- [16] T. Geppert, S. Deml, D. Sturzenegger, and N. Ebert, “Trusted Execution Environments: Applications and Organizational Challenges,” *Frontiers in Computer Science*, vol. 4, 2022. [Online]. Available: <https://www.frontiersin.org/articles/10.3389/fcomp.2022.930741> [Page 10.]
- [17] “Intel® Software Guard Extensions.” [Online]. Available: <https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html> [Page 10.]
- [18] “AMD Secure Encrypted Virtualization (SEV).” [Online]. Available: <https://www.amd.com/en/developer/sev.html> [Page 10.]
- [19] A. Ltd, “Cortex-A TrustZone – Arm®.” [Online]. Available: <https://www.arm.com/ja/technologies/trustzone-for-cortex-a> [Page 10.]
- [20] D. Kaplan, “AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More.” [Pages 10 and 11.]
- [21] R. Guanciale, N. Paladi, and A. Vahidi, “SoK: Confidential Quartet - Comparison of Platforms for Virtualization-Based Confidential Computing,” in *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*, Sep. 2022. doi: 10.1109/SEED55351.2022.00017 pp. 109–120. [Pages 10 and 11.]
- [22] M. Li, Y. Zhang, K. Li, Y. Cheng, and H. Wang, “CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel.” [Page 11.]
- [23] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, “A Survey on Homomorphic Encryption Schemes: Theory and Implementation,” *ACM Computing Surveys*, vol. 51, no. 4, pp. 79:1–79:35, 2018. doi: 10.1145/3214303. [Online]. Available: <https://doi.org/10.1145/3214303> [Pages 12 and 13.]
- [24] R. VF and L. J, “An Overview on Homomorphic Encryption Algorithms,” 2018. [Online]. Available: <https://www.ic.unicamp.br/~reltech/PFG/2018/PFG-18-28.pdf> [Page 12.]

- [25] M. Babenko, A. Tchernykh, E. Golimblevskaia, L. B. Pulido-Gaytan, and A. Avetisyan, “Homomorphic Comparison Methods: Technologies, Challenges, and Opportunities,” in *2020 International Conference Engineering and Telecommunication (En&T)*, Nov. 2020. doi: 10.1109/EnT50437.2020.9431252 pp. 1–5. [Page 12.]
- [26] C. Gentry, S. Halevi, and N. P. Smart, “Better Bootstrapping in Fully Homomorphic Encryption,” in *Public Key Cryptography – PKC 2012*, ser. Lecture Notes in Computer Science, M. Fischlin, J. Buchmann, and M. Manulis, Eds. Berlin, Heidelberg: Springer, 2012. doi: 10.1007/978-3-642-30057-8<sub>1</sub>. ISBN 978 – 3 – 642 – 30057 – 8 pp. 1 – 16. [Page 12.]
- [27] J. Fan and F. Vercauteren, “Somewhat Practical Fully Homomorphic Encryption,” 2012, report Number: 144. [Online]. Available: <https://eprint.iacr.org/2012/144> [Pages 12, 41, and 46.]
- [28] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic Encryption for Arithmetic of Approximate Numbers,” in *Advances in Cryptology – ASIACRYPT 2017*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds. Cham: Springer International Publishing, 2017. doi: 10.1007/978-3-319-70694-8<sub>15</sub>. ISBN 978 – 3 – 319 – 70694 – 8 pp. 409 – 437. [Pages 12, 41, and 46.]
- [29] J.-W. Lee, E. Lee, Y. Lee, Y.-S. Kim, and J.-S. No, “High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function,” in *Advances in Cryptology – EUROCRYPT 2021*, ser. Lecture Notes in Computer Science, A. Canteaut and F.-X. Standaert, Eds. Cham: Springer International Publishing, 2021. doi: 10.1007/978-3-030-77870-5<sub>2</sub>. ISBN 978 – 3 – 030 – 77870 – 5 pp. 618 – 647. [Page 12.]
- [30] E. Lee, J.-W. Lee, Y.-S. Kim, and J.-S. No, “Optimization of Homomorphic Comparison Algorithm on RNS-CKKS Scheme,” 2021, report Number: 1215. [Online]. Available: <https://eprint.iacr.org/2021/1215> [Page 12.]
- [31] P. Paillier, “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,” in *Advances in Cryptology — EUROCRYPT ’99*, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, vol. 1592, pp. 223–238. ISBN 978-3-540-65889-4 Series Title: Lecture Notes in Computer Science. [Online]. Available: [http://link.springer.com/10.1007/3-540-48910-X\\_16](http://link.springer.com/10.1007/3-540-48910-X_16) [Page 12.]

- [32] M. Gaid and S. Salloum, "Homomorphic Encryption," May 2021. doi: 10.1007/978-3-030-76346-6<sub>5</sub>6. ISBN 978 – 3 – 030 – 76345 – 9 [Page 13.]
- [33] A. Bhattacharya, "Homomorphic Encryption - Basics," Dec. 2020. [Online]. Available: <https://34.237.61.189/introduction-to-homomorphic-encryption/> [Page 13.]
- [34] M. Kara, A. Laouid, M. Hammoudeh, A. Bounceur, and L. Laboratory, "One Digit Checksum for Data Integrity Verification of Cloud-executed Homomorphic Encryption Operations." [Page 13.]
- [35] M. Aledhari, R. Razzak, R. M. Parizi, and F. Saeed, "Federated Learning: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Access*, vol. 8, pp. 140 699–140 725, 2020. doi: 10.1109/ACCESS.2020.3013541 Conference Name: IEEE Access. [Pages 14, 15, and 16.]
- [36] N. Rieke, J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein, S. Ourselin, M. Sheller, R. M. Summers, A. Trask, D. Xu, M. Baust, and M. J. Cardoso, "The future of digital health with federated learning," *npj Digital Medicine*, vol. 3, no. 1, pp. 1–7, Sep. 2020. doi: 10.1038/s41746-020-00323-1 Number: 1 Publisher: Nature Publishing Group. [Online]. Available: <https://www.nature.com/articles/s41746-020-00323-1> [Page 15.]
- [37] A. Alferaidi, K. Yadav, Y. Alharbi, W. Viriyasitavat, S. Kautish, and G. Dhiman, "Federated Learning Algorithms to Optimize the Client and Cost Selections," *Mathematical Problems in Engineering*, vol. 2022, p. e8514562, Apr. 2022. doi: 10.1155/2022/8514562 Publisher: Hindawi. [Online]. Available: <https://www.hindawi.com/journals/mpe/2022/8514562/> [Page 15.]
- [38] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," Jan. 2023, arXiv:1602.05629 [cs]. [Online]. Available: <http://arxiv.org/abs/1602.05629> [Page 15.]
- [39] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," Nov. 2017, arXiv:1711.10677 [cs]. [Online]. Available: <http://arxiv.org/abs/1711.10677> [Page 15.]

- [40] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, “FedHealth: A Federated Transfer Learning Framework for Wearable Healthcare,” *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 83–93, Jul. 2020. doi: 10.1109/MIS.2020.2988604 Conference Name: IEEE Intelligent Systems. [Page 15.]
- [41] J. Qi, Q. Zhou, L. Lei, and K. Zheng, “Federated Reinforcement Learning: Techniques, Applications, and Open Challenges,” Oct. 2021, arXiv:2108.11887 [cs]. [Online]. Available: <http://arxiv.org/abs/2108.11887> [Page 15.]
- [42] H. Wang, Z. Kaplan, D. Niu, and B. Li, “Optimizing Federated Learning on Non-IID Data with Reinforcement Learning,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, Jul. 2020. doi: 10.1109/INFOCOM41043.2020.9155494 pp. 1698–1707, iSSN: 2641-9874. [Page 15.]
- [43] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated Learning: Strategies for Improving Communication Efficiency,” Oct. 2017, arXiv:1610.05492 [cs]. [Online]. Available: <http://arxiv.org/abs/1610.05492> [Page 15.]
- [44] G. A. Kaissis, M. R. Makowski, D. Rückert, and R. F. Braren, “Secure, privacy-preserving and federated machine learning in medical imaging,” *Nature Machine Intelligence*, vol. 2, no. 6, pp. 305–311, Jun. 2020. doi: 10.1038/s42256-020-0186-1 Number: 6 Publisher: Nature Publishing Group. [Online]. Available: <https://www.nature.com/articles/s42256-020-0186-1> [Page 15.]
- [45] D. Froelicher, J. R. Troncoso-Pastoriza, J. L. Raisaro, M. A. Cuendet, J. S. Sousa, H. Cho, B. Berger, J. Fellay, and J.-P. Hubaux, “Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption,” *Nature Communications*, vol. 12, no. 1, p. 5910, Oct. 2021. doi: 10.1038/s41467-021-25972-y Number: 1 Publisher: Nature Publishing Group. [Online]. Available: <https://www.nature.com/articles/s41467-021-25972-y> [Pages 15 and 25.]
- [46] “ML | Stochastic Gradient Descent (SGD),” Feb. 2019, section: Computer Subject. [Online]. Available: <https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/> [Page 16.]
- [47] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated Optimization in Heterogeneous Networks,”

- Apr. 2020, arXiv:1812.06127 [cs, stat]. [Online]. Available: <http://arxiv.org/abs/1812.06127> [Pages 16 and 41.]
- [48] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership Inference Attacks against Machine Learning Models,” Mar. 2017, arXiv:1610.05820 [cs, stat]. [Online]. Available: <http://arxiv.org/abs/1610.05820> [Page 17.]
- [49] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, “PPFL: privacy-preserving federated learning with trusted execution environments,” in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '21. New York, NY, USA: Association for Computing Machinery, 2021. doi: 10.1145/3458864.3466628. ISBN 978-1-4503-8443-8 pp. 94–108. [Online]. Available: <https://doi.org/10.1145/3458864.3466628> [Page 17.]
- [50] J. Park and H. Lim, “Privacy-Preserving Federated Learning Using Homomorphic Encryption,” *Applied Sciences*, vol. 12, no. 2, p. 734, Jan. 2022. doi: 10.3390/app12020734 Number: 2 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: <https://www.mdpi.com/2076-3417/12/2/734> [Pages 17 and 26.]
- [51] M. Mansouri, M. Önen, W. Ben Jaballah, and M. Conti, “SoK: Secure Aggregation Based on Cryptographic Schemes for Federated Learning,” *Proceedings on Privacy Enhancing Technologies*, vol. 2023, no. 1, pp. 140–157, Jan. 2023. doi: 10.56553/popets-2023-0009. [Online]. Available: <https://petsymposium.org/popets/2023/popets-2023-0009.php> [Page 17.]
- [52] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical Secure Aggregation for Privacy-Preserving Machine Learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017. doi: 10.1145/3133956.3133982. ISBN 978-1-4503-4946-8 pp. 1175–1191. [Online]. Available: <https://doi.org/10.1145/3133956.3133982> [Pages 17, 18, and 24.]
- [53] H. Zhu, R. Wang, Y. Jin, K. Liang, and J. Ning, “Distributed additive encryption and quantization for privacy preserving federated deep learning,” *Neurocomputing*, vol. 463, pp. 309–327, Nov. 2021. doi: 10.1016/j.neucom.2021.08.062. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231221012522> [Page 17.]

- [54] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, “Multiparty Homomorphic Encryption from Ring-Learning-With-Errors.” [Online]. Available: <https://eprint.iacr.org/undefined/undefined> [Page 17.]
- [55] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, “A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection,” *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021. doi: 10.1109/TKDE.2021.3124599 Conference Name: IEEE Transactions on Knowledge and Data Engineering. [Page 18.]
- [56] “PySyft,” Mar. 2023, original-date: 2017-07-18T20:41:16Z. [Online]. Available: <https://github.com/OpenMined/PySyft> [Page 18.]
- [57] “IBM Federated Learning,” Mar. 2023, original-date: 2020-06-19T17:19:21Z. [Online]. Available: <https://github.com/IBM/federated-learning-lib> [Page 18.]
- [58] “Open Federated Learning (OpenFL) - An Open-Source Framework For Federated Learning,” Mar. 2023, original-date: 2021-01-12T21:29:52Z. [Online]. Available: <https://github.com/securefederatedai/openfl> [Page 18.]
- [59] Z. Jiang, “FLASHE,” Feb. 2023, original-date: 2021-06-17T05:05:30Z. [Online]. Available: <https://github.com/SamuelGong/FLASHE> [Pages 19 and 33.]
- [60] “tf-encrypted/tf-encrypted: A Framework for Encrypted Machine Learning in TensorFlow.” [Online]. Available: <https://github.com/tf-encrypted/tf-encrypted> [Page 19.]
- [61] “TF SEAL,” Mar. 2023, original-date: 2019-07-08T12:28:53Z. [Online]. Available: <https://github.com/tf-encrypted/tf-seal> [Page 19.]
- [62] “TenSEAL,” Mar. 2023, original-date: 2020-01-25T14:36:55Z. [Online]. Available: <https://github.com/OpenMined/TenSEAL> [Page 19.]
- [63] “Pyfhel 3.4.1 documentation.” [Online]. Available: <https://pyfhel.readthedocs.io/en/latest/> [Pages 19, 41, and 54.]
- [64] H. Li, “Salvia: Secure Aggregation for Federated Learning in Flower.” [Online]. Available: <https://hei411.github.io/projects/salvia.html> [Page 19.]
- [65] K. H. Li, P. P. B. de Gusmão, D. J. Beutel, and N. D. Lane, “Secure aggregation for federated learning in flower,” in *Proceedings of the 2nd ACM International*

- Workshop on Distributed Machine Learning*. Virtual Event Germany: ACM, Dec. 2021. doi: 10.1145/3488659.3493776. ISBN 978-1-4503-9134-4 pp. 8–14. [Online]. Available: <https://dl.acm.org/doi/10.1145/3488659.3493776> [Page 19.]
- [66] “The SSL Protocol.” [Online]. Available: <http://www.webstart.com/jed/papers/HRM/references/ssl.html> [Page 20.]
- [67] C. Allen and T. Dierks, “The TLS Protocol Version 1.0,” Internet Engineering Task Force, Request for Comments RFC 2246, Jan. 1999, num Pages: 80. [Online]. Available: <https://datatracker.ietf.org/doc/rfc2246> [Page 20.]
- [68] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” Internet Engineering Task Force, Request for Comments RFC 8446, Aug. 2018, num Pages: 160. [Online]. Available: <https://datatracker.ietf.org/doc/rfc8446> [Page 21.]
- [69] “TLS Security 5: Establishing a TLS Connection,” Mar. 2019. [Online]. Available: <https://www.acunetix.com/blog/articles/establishing-tls-ssl-connection-part-5/> [Page 21.]
- [70] E. Rescorla, “HTTP Over TLS,” Internet Engineering Task Force, Request for Comments RFC 2818, May 2000, num Pages: 7. [Online]. Available: <https://datatracker.ietf.org/doc/rfc2818> [Page 21.]
- [71] C. Newman, “Using TLS with IMAP, POP3 and ACAP,” Internet Engineering Task Force, Request for Comments RFC 2595, Jun. 1999, num Pages: 15. [Online]. Available: <https://datatracker.ietf.org/doc/rfc2595> [Page 21.]
- [72] W. Tounsi and H. Rais, “A survey on technical threat intelligence in the age of sophisticated cyber attacks,” *Computers & Security*, vol. 72, pp. 212–233, Jan. 2018. doi: 10.1016/j.cose.2017.09.001. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404817301839> [Pages 21 and 23.]
- [73] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, “Cyber Threat Intelligence Sharing Scheme Based on Federated Learning for Network Intrusion Detection,” *Journal of Network and Systems Management*, vol. 31, no. 1, p. 3, Oct. 2022. doi: 10.1007/s10922-022-09691-3. [Online]. Available: <https://doi.org/10.1007/s10922-022-09691-3> [Pages 21, 22, 23, and 26.]
- [74] N. Oliveira, I. Praça, E. Maia, and O. Sousa, “Intelligent Cyber Attack Detection and Classification for Network-Based Intrusion Detection

- Systems,” *Applied Sciences*, vol. 11, no. 4, p. 1674, Jan. 2021. doi: 10.3390/app11041674 Number: 4 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: <https://www.mdpi.com/2076-3417/11/4/1674> [Page 22.]
- [75] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, “Secure Single-Server Aggregation with (Poly)Logarithmic Overhead,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. Virtual Event USA: ACM, Oct. 2020. doi: 10.1145/3372297.3417885. ISBN 978-1-4503-7089-9 pp. 1253–1269. [Online]. Available: <https://dl.acm.org/doi/10.1145/3372297.3417885> [Page 24.]
- [76] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. Vincent Poor, “Federated Learning With Differential Privacy: Algorithms and Performance Analysis,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020. doi: 10.1109/TIFS.2020.2988575 Conference Name: IEEE Transactions on Information Forensics and Security. [Page 24.]
- [77] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, “{BatchCrypt}: Efficient Homomorphic Encryption for {Cross-Silo} Federated Learning,” 2020. ISBN 978-1-939133-14-4 pp. 493–506. [Online]. Available: <https://www.usenix.org/conference/atc20/presentation/zhang-chengliang> [Pages 25 and 47.]
- [78] J. Ma, S. Naas, S. Sigg, and X. Lyu, “Privacy-preserving federated learning based on multi-key homomorphic encryption,” *International Journal of Intelligent Systems*, vol. 37, no. 9, pp. 5880–5901, Sep. 2022. doi: 10.1002/int.22818. [Online]. Available: <https://onlinelibrary.wiley.com/doi/10.1002/int.22818> [Page 26.]
- [79] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, “Data Poisoning Attacks Against Federated Learning Systems,” in *Computer Security – ESORICS 2020*, ser. Lecture Notes in Computer Science, L. Chen, N. Li, K. Liang, and S. Schneider, Eds. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-58951-6\_24. ISBN 978 – 3 – 030 – 58951 – 6 pp. 480 – 501. [Page 29.]
- [80] C. Fung, C. J. M. Yoon, and I. Beschastnikh, “Mitigating Sybils in Federated Learning Poisoning,” Jul. 2020, arXiv:1808.04866 [cs, stat]. [Online]. Available: <http://arxiv.org/abs/1808.04866> [Page 29.]

- [81] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, F. Piessens, and D. Gruss, “Plundervolt: How a Little Bit of Undervolting Can Create a Lot of Trouble,” *IEEE Security & Privacy*, vol. 18, no. 5, pp. 28–37, Sep. 2020. doi: 10.1109/MSEC.2020.2990495 Conference Name: IEEE Security & Privacy. [Page 31.]
- [82] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, “Flower: A Friendly Federated Learning Research Framework,” Mar. 2022, arXiv:2007.14390 [cs, stat]. [Online]. Available: <http://arxiv.org/abs/2007.14390> [Page 32.]
- [83] “Pipenv: Python Dev Workflow for Humans — pipenv 2023.6.12.dev0 documentation.” [Online]. Available: <https://pipenv.pypa.io/en/latest/> [Page 32.]
- [84] L. Deng, “The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web],” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, Nov. 2012. doi: 10.1109/MSP.2012.2211477 Conference Name: IEEE Signal Processing Magazine. [Page 35.]
- [85] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep Learning with Limited Numerical Precision.” [Page 48.]

