Privacy Preservation and Verifiability for Federated Learning

By Jianxiang Zhao

A thesis submitted to the Graduate Program in Electrical And Computer Engineering in conformity with the requirements for the Degree of Master of Applied Science

Queen's University
Kingston, Ontario, Canada
January, 2023

Copyright © Jianxiang Zhao, 2023

Abstract

Federated learning is a distributed machine learning framework to address the bottleneck of traditional machine learning on data collection and privacy leakage, which allows training a learning model using distributedly stored data without exposing them. In federated learning, multiple clients collaboratively train a single global model that is improved iteratively through clients' local data. Each client receives the global model from an aggregation server. Then, they train it on their private data, and send locally trained models back to the server, which are later integrated into the global model. Iteration after iteration, the federated training continues until the global model is considered well-trained.

Federated learning provides basic privacy protection against outsider attackers, but it does not mean user privacy would not be leaked. It has been proved that the local models shared with the server are vulnerable to leaking the raw data maintained by users. To preserve the privacy of users, shared models shall not be in the form of plaintext. However, the encryption of the local models in sharing brings the challenge of model aggregation for the server, which is important to ensure fast convergence of the global model. In addition, it is hard to ensure that the server could honestly aggregate the local models, especially in cross-silo federated learning. If the server does not have enough motivation to coordinate the training, the performance of federated

learning cannot be guaranteed.

In this thesis, we aim to prevent user privacy leakage from the shared local models, and guarantee the correctness of the global models output by the server. Specifically, we first propose PPA-AFL, a fully asynchronous secure federated learning protocol, which addresses the privacy issue in the asynchronous federated aggregation. The disadvantage of PPA-AFL is that it requires two non-colluding servers and cannot provide a correctness guarantee for the global model. Then, we design PPVA-AFL, a secure and verifiable aggregation protocol for asynchronous federated learning, which simultaneously guarantees the privacy of the local model and the correctness of the global model. In short, we have investigated security issues in federated learning and developed two novel schemes that enhance privacy preservation and introduce verifiability for federated learning.

Acknowledgments

First and foremost I am extremely grateful to my supervisor, Prof. Jianbing Ni for his invaluable advice, continuous support, and patience during my study. His immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life.

I am also thankful to the department of electrical and computer engineering and all its staff for all the considerate guidance. They have stimulated my learning through lectures and discussions. I must thank my student colleagues, whose passion for their own research has inspired me to work harder and strive to produce the highest quality of research.

In addition, I would like to mention that I am deeply indebted to anyone who has directly assisted my thesis, be it through discussions, insightful comments or proofreading this document.

Finally, I would like to express my gratitude to my family and my friends. Without their tremendous understanding and encouragement over the past few years, it would be impossible for me to complete my study.

Statement Of Originality

The following works are my own and I hereby certify the intellectual content of this thesis is the product of my own work. All references and contributions of other individuals have been cited and sourced appropriately, as defined by the IEEE Citation Reference manual.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Traditional, the computer can only follow the detailed instruction given by the programmer. The appearance of machine learning changes this paradigm. Machine learning is data-driven and algorithm-driven. When giving a "framework program" data and letting it learn the regular patterns from it, a model with the ability to solve similar problems will be produced. Machine learning has been implemented in various industries. The adoption of the machine learning technique simplifies work content and improves efficiency. The machine learning based solution demonstrates the ability to be as good as or even better than human work in more fields. Currently, practical face recognition [1] and self-driving [2] are powered by machine learning. In addition, in medical diagnosis [3] and financial fraud detection [4], machine learning greatly improved accuracy and efficiency.

With the rapid development of machine learning algorithms and hardware, the current bottleneck is often the shortage of data [5]. A situation exists where someone wants to train a machine learning model but the required data is not acquired. There

are two reasons for the shortage of data. One is that some data are generated on distributed devices and data transmission can be costly, such as records on personal mobile devices. The other is that some data related to personally sensitive information is not allowed to be used directly due to the privacy protection regulations, such as GDPR [6]. Federated learning has been proposed to utilize data stored at different places [7], that is, a global model is trained from the data that are distributively maintained on different devices, instead of the data on a centralized server. The data owner can contribute to a global model without sharing their local data by participating in federated training.

Federated learning involves two kinds of entities, a server and clients as follows:

- The server: The server is a relatively powerful machine that has high computing capability, reliable network connection, and large storage. The server does not have data for the training tasks. The goal of the server is to produce a global model. This server can be the initiator of federated learning that has the motivation to train a model for prediction or hired by the clients to serve as a coordinator for the training process.

- The client: A client can be a device or an organization with the local dataset. Generally, a client does not have or does not need to have high computing ability, reliable network connection, and large storage. A client can find benefits in the deployment of a global model, which are the incentives for joining federated training. Clients can join voluntarily when finding the opportunity on the Internet, or can be invited by the server.

According to the training tasks and the specific roles of clients, there are two settings of federated learning, cross-silo and cross-device.

- Cross-silo federated learning: In the cross-silo setting, clients are organizations who have local data, but none of their data is diverse enough. To train a model that has good generalization ability, they initiate federated learning together. The topology of the system is illustrated in Figure 1.1. The server can be provided through a server renting service from a trusted company or institution. For example, a federated learning server can be implemented on Google Cloud [8] and IBM Cloud [9]. The clients may meet the hardware requirement to train the model but the lack of data encourages them to take a part in federated learning. In this setting, the clients are often fixed. It is likely that all clients can participate in the training from the beginning to the end, and they contribute relatively equally. The ownership of the final model belongs to the clients who contribute to the training. The Cross-Silo Federated Learning has been taken into practice in the medicare industry [10], government statistics [11], etc.

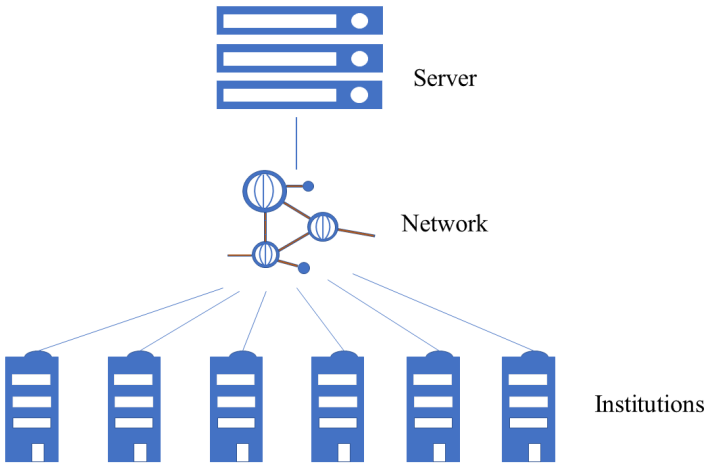

Figure 1.1: Cross-Silo Federated Learning

- Cross-device federated learning: In the cross-device setting, clients are devices

that have local data, but the small amount of data and the low computing capability hinder them from training a usable model. The topology of the system is illustrated in Figure 1.2. The devices are often "heterogeneous" and "weak". "Heterogeneous" means they vary in computing abilities and network latency. "Weak" means they have a disadvantage in computing resources, communication bandwidth, storage, and battery life. In this setting, federated learning is commonly initialized by the server. The owner of the server has the motivation to train a model, but the data is distributed. This kind of federated training is often "open", which means the system welcomes everyone and clients can join the training when they are available. For clients, the training is occasional, so that they can participate only during some periods. No matter how much a client has contributed to the training, the final model will benefit them somehow, which becomes the main motivation for joining the federated training. This type of federated learning is commonly adopted in IoT projects [12] and recommendation systems [13].

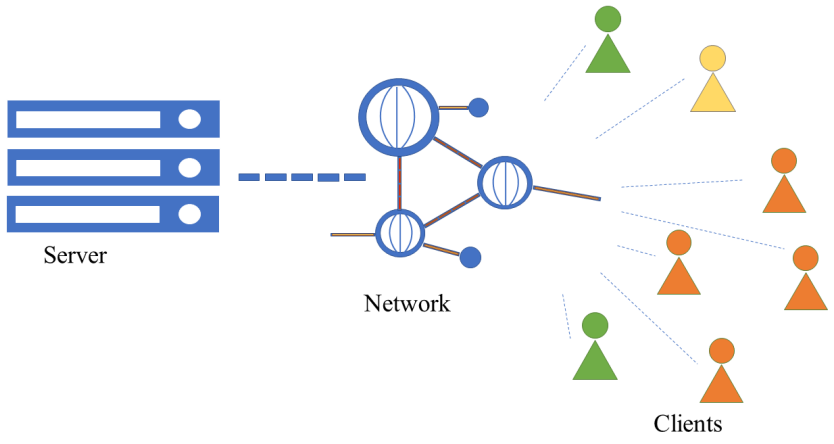

Figure 1.2: Cross-Device Federated Learning

Adopting cross-silo or cross-device settings is determined by the task itself. From the view of implementation, the cross-silo setting can be deemed as a special case of cross-device settings, in which there is less restriction on the hardware and network accessibility of clients. Most research findings in cross-device settings can apply in the cross-silo setting but not vice versa.

Federated learning produces a model through cooperative training. The global models are aggregated every time a group of clients submit their local models. Using the cross-device setting as an example, the life cycle of a federated model is as follows:

- Data collection: Data used in federated learning is collected by each client independently. Generally, It is assumed that clients take samples from the same distribution. Because of the different geographic locations, different objects of interest, and different accessible resources, the single local dataset is often biased. To train an unbiased model, multiple datasets of different clients are needed. In practice, raw data cannot depart from its own for some reasons. For example, some IoT devices have very limited bandwidth and battery life [14], so that they are not suitable for transferring the dataset. In addition, the personalized data collected on users' devices cannot be requested directly due to their concerns about privacy leakage or the privacy regulations like GDPR [15]. Therefore, the collected data is kept on the device and not shared with other clients or the server during training.

- Model training: There are two different models in federated learning. One is the local model, which is trained by clients, based on the latest global model and using the local data on devices. The other is the global model, which is the product of the aggregation algorithm. Aggregation takes a group of

clients' local models as input and outputs a new global model. The training workflow is illustrated in Figure 1.3, in which one client is taken as an example (other clients behave the same) to show how the local training and global model aggregation happen in one round of the federated training. The first global model is randomly generated and later ones are aggregated from local models. When a new round of training begins, clients get the latest global model and train the new local model based on the global model using their local data. The local models are sent to the server as an update. This process happens repeatedly until the global model is considered well-trained.

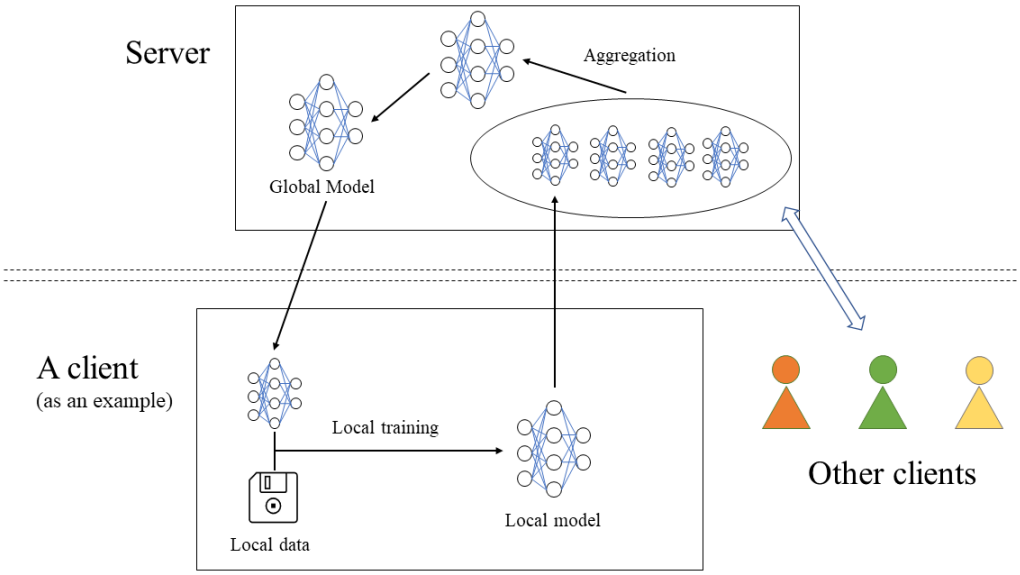

Figure 1.3: Federated Training

- Model validation: It is assumed that clients take samples from the same distribution. The global model is expected to perform well in all situations that a client may meet. There are two methods to validate the global model. The

first one is that the server collects a small amount of various unbiased data and uses it as a validation set to test the model. In practice, unbiased data is often collected from clients if clients are willing to share a little of their data. The other approach is that let clients compare the global model and their local model. If the global model performs better than local models, it is considered that federated learning is useful in this task.

- Model deployment: After the validation, a well-trained model can be deployed on devices. Not all devices contribute to the global model but the global model can enhance the functionality of all similar devices. For example, Google uses federated learning to improve the prediction performance of Google keyboard [16].

Federated learning enables cooperative training when the data cannot depart from the original owner. Compare to the traditional solution, it reduces the overhead of data transfer and provides basic privacy protection.

1.2 Motivation

1.2.1 Privacy

Federated learning allows multiple parties to train a model together and their data never depart from their devices. However, privacy leakage can still happen even if the raw data is not exposed [17]. In federated learning, clients use local models or local gradients as the update, which are sent to the server. The privacy concern concentrates on how much sensitive information can be exploited from clients' updates. In traditional machine learning, there are several known attacks on the trained

model. For example, model reverse attack [18] and membership inference attack [19] can recover parts of sensitive information of the raw data from the trained models. In federated learning, these attacks are still effective to learn sensitive information about raw data from clients' updates [20]. Keeping data locally does not mean privacy. The original federated learning has the hazard of privacy leakage. Therefore, it is critical to protect the clients' updates for preserving their privacy. [21].

1.2.2 Efficiency

Cross-device federated learning has the "device heterogeneity" assumption, which means devices vary in computing capability, communication capability, etc. In addition, cross-device federated learning is often an "open" system that attracts clients to join in the training through the Internet [22]. A small number of poorly performing devices can drag down the efficiency of the overall system [23]. Federated learning usually takes more rounds to generate a usable model because it suffers from unbalanced client selections and poor updates [24]. The slow convergence of federated learning and high latency aggregation makes the whole system lack efficiency. The workflow of the original federated learning shall be changed to improve efficiency when clients create and submit updates slowly.

1.2.3 Verifiability

The models trained on the client side are local models, and the final output of the system is the global model. Clients expect the global model can perform better than their local model, which is the reason for participating in the federated training. A risk exists that the server becomes a "lazy" server. Such a server does not perform the

aggregation honestly. It may only use a small subset of updates to do the aggregation or even do not perform the aggregation to save the cost. For example, after several rounds of training, the server stops doing the aggregation but adds some random noise to the global model, and claims the new global model is honestly generated. Dishonest servers damage the interests of clients, but the original federated learning cannot prevent this threat. Therefore, it is meaningful to achieve the verifiability of the global model to make sure that the global model, which will incentivize user participation. Verifiable computation can force the entrusted task to be done following the protocols [25, 26].

1.3 Problem Statement

The above features are not straightforward to be achieved at the same time. There is always a trade-off between security properties and the system performance goals. Privacy-preserving and verifiability have a natural conflict because the most common method to verify something is to inspect it in the cleartext but privacy protection aims to hinder that. Therefore, our goal is to achieve both privacy-preserving and verifiability at the same time, while controlling the performance overhead in an acceptable range.

The privacy-preserving feature is relatively easy to achieve when verifiability is not considered. The updates can be hidden by randomness or can be encrypted to prevent privacy leakage from the updates. Verifiability is also achievable if the operation of a server is visible to clients in real-time. Combining these properties together requires the adoption of several cryptographic preliminaries and the redesign of the federated learning protocol. That is, the original federated learning needs to be disassembled,

added with cryptographic techniques, and reassembled into a new protocol.

To achieve better efficiency, the original federated learning should be modified into an asynchronous alternative, in which clients can perform the local training and submit updates at any given time without waiting for other parties. This affects the functionality of some cryptographic primitives and the convergence performance of the model. Adopting asynchronous workflow while keeping security properties requires more consideration on how to use cryptographic primitives in the protocols. In addition, due to the asynchronous setting, the convergence performance also needs to be analyzed.

We formalize our research questions as follows:

- How to preserve the privacy of clients' local updates for asynchronous aggregation in federated learning?
- How to enable clients to perform the verifiability of local update aggregation, without exposing the local updates in asynchronous federated learning?
- How to guarantee the convergence performance for federated learning adopting asynchronous aggregation?

1.4 Thesis Contributions

We aim to achieve privacy-preserving asynchronous federated learning that preserves the privacy of clients' locate updates in asynchronous aggregation, while enabling clients to perform the verifiability of local update aggregation. Specifically, the detailed contributions can be summarized in two-fold:

- Privacy-Preserving Model Aggregation for Asynchronous Federated Learning:

  We propose a privacy-preserving model aggregation for asynchronous federated learning (PPA-AFL) to allow clients to asynchronously participate in the federated learning and address the privacy leakage concerns for clients. In PPA-AFL, clients can determine when to participate in the training. When a client is available, it can send a request to get the auxiliary information for the training and create an update. The training and submitting processes are fully asynchronous. To protect the client's updates and support local model aggregation, we utilize the Paillier encryption [27] to encrypt the local update and support homomorphic aggregation. Also, the secret sharing scheme is utilized to support the sharing of the decryption key and enable privacy-preserving asynchronous aggregation. In doing so, the server is not able to learn any information about the local updates, but can asynchronously aggregate to produce the global model. Finally, we show the efficiency of our proposed protocol through complexity analysis and extensive experiments on a prototype implementation.

- Privacy-preserving and Verifiable Buffered Asynchronous Federated Learning:

  We propose a privacy-preserving and verifiable buffered asynchronous federated learning (PPVA-AFL) to enable both the verifiability of the local model aggregation and the privacy preservation of clients' local updates in asynchronous federated learning. In PPVA-AFL, we adopt buffered asynchronous federated aggregation with staleness function [28] to allow the server to aggregate the local updates that come from clients at different times. The protocol loose the

synchronization requirement compared with other privacy-preserving federated learning protocols [29, 30]. The local models of the clients are protected by using the random noises against the server, other clients, and potential attackers, while enabling the server to generate the global model by aggregating buffered local models. The secret sharing scheme is utilized by the clients to guarantee that the random noise added by each client in the local model can be removed in the global model after aggregation. Verifiable computation and commitment protocols adopted in our design force the server to aggregate the local models in its buffer honestly. We demonstrate the efficiency of our PPVA-AFL through complexity analysis and extensive experiments and test the convergence performance through our prototype implementation.

1.5 Thesis Outline

The structure of the thesis is as follows. In Chapter 2, we introduce the background of this thesis, including the cryptographic primitives and the related work in federated learning, such as privacy-preserving federated learning, verifiable federated learning, and asynchronous federated learning. In Chapter 3, we propose our PPA-AFL, including a system model, detailed design, and security and performance analysis. In Chapter 4, we propose our PPVA-AFL, including a system model, detailed design, and security and performance analysis. In Chapter 5, we summarize the thesis and present the future research problems, including blockchain-based federated learning, Byzantine-resistant federated learning, personalized federated learning, and incentive mechanism for federated learning.

Chapter 2

Background

In this chapter, we review the cryptographic primitives used in this thesis, including public key encryption, homomorphic encryption, secret sharing, key agreement, homomorphic hash function, and commitment, and present the related work, including privacy-preserving federated learning, verifiable federated learning, and asynchronous federated learning.

2.1 Cryptographic Preliminaries

2.1.1 Key Agreement

Diffie–Hellman key exchange is one of the widely used key agreement schemes. It is an interactive scheme, and the detailed processes are as follows:

- $Setup(1^k)$

  This setup method gives public parameters as $Param \leftarrow Setup(1^k)$. This $Param$ is implicitly used in the following algorithms for simplicity.

- $KeyGen(1^k)$

  A pair of keys is generated upon calling this method, as $(sk, pk) \leftarrow KeyGen()$.

pk is the public key and sk is the secret key.

- $Agree(sk_i, pk_j)$

  When using the public key pk_j of user j, user i can input its secret key sk_i at the same time to get the agreed key $ak_{i,j} \leftarrow Agree(sk_i, pk_j)$.

2.1.2 Secret Sharing

Shamir's secret sharing is one of the widely used secret sharing schemes, in which the generated shares have additive homomorphism. It consists of the following three algorithms:

- $Setup(1^k)$

  On inputting the security parameter, the algorithm gives $Param \leftarrow Setup(1^k)$. $Param$ is the parameter for the secret sharing protocol and is implicitly used in the following algorithms.

- $Share(n, t, m)$

  There is a pre-determined number of shares n and a recovery threshold t, and a message m. Call the algorithm to get $\{[s_i]\}_{i\in[0,n)} \leftarrow Share(n, t, m)$. $\{[s_i]\}_{i\in[0,n)}$ are n shares.

- $Combine(t, \{[s_i]\}_{i\in P'})$

  There is $\{[s_i]\}_{i\in P} \leftarrow Share(n, t, m)$. To be noted, P is a set of virtual parties to hold the shares and $|P| = n$. P' is a subset of P, in additional $|P'| > t$. The combination algorithm outputs the message as $m \leftarrow Combine(t, \{[s_i]\}_{i\in P'})$

2.1.3 Symmetric Encryption

Symmetric encryption consists of the following three algorithms:

- $KeyGen(1^p)$

  Given the security parameter p, the key generation algorithm outputs a symmetric key k, as $k \leftarrow KeyGen(1^p)$.

- $Enc(m, k)$

  The algorithm performs the encryption on the plaintext m with the key k, and outputs ciphertext $c \leftarrow Enc(m, k)$.

- $Dec(c, k)$

  The algorithm performs the decryption on the ciphertext c using the possible key k. When the decryption is successful, it outputs $m \leftarrow Dec(c, k)$; otherwise, it rises an error.

2.1.4 Public Key Encryption

ElGamal encryption is one of the widely used public key encryption schemes. It consists of the following four algorithms:

- $Setup(1^k)$

  This setup method gives public parameters as $Param \leftarrow Setup(1^k)$. This $Param$ is implicitly used in the following algorithms for simplicity.

- $KeyGen(Param)$

  A pair of keys is generated upon calling this method, as $(sk, pk) \leftarrow KeyGen(Param)$. pk is the public key and sk is the secret key.

- $Enc(pk, m)$

  When giving the message m and the public key pk, the encryption is $c \leftarrow Enc(pk, m)$. c is the ciphertext of m.

- $Dec(sk, c)$

  Given the ciphertext c and the corresponding secret key sk, the decryption is $m \leftarrow Dec(sk, c)$.

2.1.5 Homomorphic Encryption

Paillier encryption is one of the widely used homomorphic encryption schemes that support additive homomorphism. It consists of the following five algorithms:

- $Setup(1^k)$

  On inputting the security parameter, the algorithm gives $Param \leftarrow Setup(1^k)$. $Param$ is the parameter for the secret sharing protocol and is implicitly used in the following algorithms.

- $Keygen(Param)$

  On inputting the $Param$ generated from $Setup(Param)$, the algorithm randomly generates a pair of keys pk and sk.

- $Enc(m, pk)$

  On inputting the public key pk and a message m, the algorithm gives $c \leftarrow Enc(m, pk)$. c is the ciphertext corresponding to the message m. The related key pair is (pk, sk).

- $Eval(\{[c_i]\}, f)$

  On inputting a set of ciphertext $[c_i]$ and a linear function f, the algorithm

gives $c_{eval} \leftarrow Eval(\{[c_i]\}, f)$. All ciphertext in $\{[c_i]\}$ is related to the same key pair (pk, sk). This evaluation should show homomorphism, which means the decryption of c_{eval} is as same as the result of feeding the decryption of $\{[c_i]\}$ to the linear function f.

- $Dec(c, sk)$

  On inputting the secret key sk and a ciphertext c, the algorithm gives $m \leftarrow Dec(c, sk)$. m is the plaintext corresponding to the ciphertext c.

2.1.6 Homomorphic Hash Function

A hash function is used to map any data of arbitrary length to its corresponding fixed-size message, which is called a hash value. Homomorphism means a mapping function can keep the structure between two algebraic structures, which means the relationship between values can be found on their mapped value in another algebraic structure. In PPVA-AFL, a generic linear homomorphic hash scheme is adopted, and the detailed processes are as follows:

- $Setup(1^k, 1^d)$

  This is the parameter generation algorithm. On taking the security parameter k and the number of data dimensions d, it outputs the public parameter $Param$. For simplicity, $Param$ is implicitly taken as an input in the $Hash$ and $Eval$.

- $Hash(m)$

  On given a message m of dimension d, $h \leftarrow Hash(m)$, which h is the hash value.

- $Eval(H, A)$

$H = \{h_1, h_2, \cdots\}$ is a set of hash values corresponding to a set of messages $\{m_1, m_2, \cdots\}$. $A = \{a_1, a_2, \cdots\}$ is a set of coefficients of a polynomial. On inputting the hash values and coefficients of a polynomial, an aggregated hash value is evaluated, as $h \leftarrow Eval(H, A)$.

2.2 Related Work

We give a brief review of the existing work on privacy-preserving federated learning, verifiable federated learning, and asynchronous federated learning.

2.2.1 Privacy Preserving Federated Learning

Federated learning provides a framework that allows training a model collaboratively without collecting the data from its original owner, but privacy concerns still exist. The local gradients and models are sent to the server as updates, which are used to calculate the new global model. The trained model and the gradients in the training process have been proven to leak the information of the training samples [31]. The model and gradient produced by a client can be analyzed to expose sensitive information that should never be shared with untrusted parties, such as the server, other clients, and potential attackers. To address the privacy leakage in the original federated learning, privacy-preserving federated learning (PPFL) has been proposed. In general, there are four methods to protect the local model and gradients: differential privacy, homomorphic encryption, trusted execution environment, and random masks.

Differential privacy (DP) [32] is designed to control information leakage for any published database and machine learning model. By adopting differential privacy,

the informational leakage can be limited to a certain level with a strong theoretical guarantee [33]. Differential privacy federated learning (DPFL) is a very effective solution for protecting local models and gradients in large-scale federated learning tasks, such as the medicare record system [34]. When adopting differential privacy, the system should balance the trade-off between privacy and performance. Adding more randomness to the updates provides stronger privacy protection but degrades the performance of the final model. Currently, the existing work on DPFL focuses on how to reduce the noise level while maintaining the same level of privacy. One approach is to change the way to add the noise. For example, instead of adding noise to the plaintext of the local model, adding noise to the ciphertext after applying some encryption [35] and adding noise to the shuffled models [36] can achieve better privacy protection with less noise. The other approach is local differential privacy (LDP) [37], which allows clients to generate noise on their local side and make the global model has the expected noise level. Differential privacy-based PPFL has a close running efficiency as the original federated learning, while the drawback is that the noise has a negative impact on the convergence during the training and the performance of the final model.

Cryptography primitives, such as homomorphic encryption (HE) [38] and secure multi-party computation (MPC) [39], can be used to protect the local models against the server in model aggregation.

Hardy et al. [40] adopted the additional homomorphic encryption to build HE-based PPFL. They found that the multiplication between plaintext and ciphertext can be deemed as repeated addition operations. Thus, they rewrite the plaintext

into matrices and apply high efficient multiplication algorithm for the matrix to reduce the overhead of HE. However, the reduced overhead is still far from acceptable. The addition operation is still 2 to 3 orders of magnitude slower than its cleartext equivalent. Liu et al. [41] applied HE to federated transfer learning. They rewrite the loss function using the polynomial approximation, because the additional homomorphic encryption is not applied to the nonlinear function like *Sigmoid*. Their privacy-preserving loss function enables collaborative transfer learning and preserves the privacy of clients. Aono et al. [42] combined the gradient update algorithm with additional homomorphic encryption. In their system, all gradients are stored at the parameter server in the form of ciphertext, which makes the server can perform the aggregation but learn nothing from the clients' updates. Zhang et al. [43] proposed BatchCrypt, an HE-based PPFL, which packs multiple plaintexts into a long integer as a batch. Therefore, many parameters can be processed in one go and the overall latency of the system decrease.

Zhu et al. [44] formalized an oracle-aided MPC solution for computing weighted aggregation in federated learning, which is secure against honest-but-curious adversaries. Kanagavelu et al. [45] proposed to develop a two-phase mechanism for applying MPC in PPFL. In the first phase, a small committee is elected from clients, then in the second phase, the committee provides MPC-enabled model aggregation service to a larger number of clients. Truex et al. [35] presented a DP-MPC hybrid alternative approach that utilizes both differential privacy and MPC to balance the trade-off between performance and efficiency. Their system is therefore a scalable approach that preserves privacy for clients in the high-efficient aggregation. Byrd et al. [46] applied DP noise to the MPC-enabled PPFL, which slightly impacts the performance

but trades for a large improvement in efficiency.

With the hardware support, the trusted execution environment (TEE) [47] can be used to build PPFL. Mondal et al. [48] described FLATEE, an efficient privacy-preserving federated learning framework across TEE, which reduces the overhead by avoiding using HE and MPC. Chen et al. [49] designed a training-integrity protocol for federated learning on the TEE, in which causative attacks can be detected. In their design, malicious participants who try to corrupt the well-learned model cannot inject false training results without being detected. Zhang et al. [50] proposed TrustFL, which uses TEE in the privacy-preserving protocol. A small fraction of all training processes are randomly checked, while all computations are executed on the co-located faster processor. They achieve high efficiency in the privacy-preserving protocol by combining it with TEE. The requirement of TEE hardware makes these approaches inconvenient to implement. This approach mainly relies on the security promise given by the hardware design [51].

To improve efficiency, mask-based PPFL has been proposed. The first practical solution is a pair mask-based PPFL proposed by Bonawitz et al. [29]. In pair mask-based PPFL, the cleartext of the local model is hidden by two different masks. The first one is generated from the key agreement protocol and the sum of masks of all clients is zero, so that the noise is cancelled out in the model aggregation. The second mask is randomly generated. The secret sharing scheme is applied to both of them to provide dropout tolerance. The original pair mask-based PPFL is an interactive protocol that has strict synchronization requirements. Clients may drop unexpectedly in any step of the protocol and cause the result incorrect if the aggregation is a weighted sum. To remove the noise, the server needs to collect the secret shares from

online clients and recover the secret for every dropped client. The drop tolerance is necessary but costly. To improve efficiency, the topology of the system and pairing logic is modified. Subsequently, some variants reduce the number of rounds and the overhead of the drop tolerance. To reduce the overhead of the mask-based secure aggregation, the masking method is improved by other works. Liu et al. [52] unitized the homomorphic property of SHPRG, a homomorphic pseudo-random generator, to simplify the masking and demasking scheme. So et al. [53] overcame the slow secret recovery for drop clients by changing from "random-seed reconstruction" to "one-shot aggregate-mask reconstruction".

2.2.2 Verifiable Federated Learning

In federated learning, there are not policies or mechanisms to guarantee that the server honestly conducts local model aggregation and global model generation. The server may not perform the calculation correctly if a bad model is still accepted by the clients. To avoid the server providing a bad model, verifiable computation has been used to check the aggregation operations of the server in federated learning [54].

Homomorphic hash [55] is commonly used in the implementation of verifiable computation. If a computation is performed as expected, the hash of the result should be consistent with the value from the same operation but take the input as hash values. In federated learning, the number of clients in a given epoch and the drop situation varies, which makes the function describing the operation cannot be pre-determined. The function is generated during aggregation. The approach of verification is to force the server to do the aggregation of hashes at the same time, and another approach is letting the server publish the function used for aggregation, which

allows clients to do the verification on their side. In both settings, if the verification passes, clients accept the result; otherwise, the result is discarded and the server may be suspected.

Xu et al. [56] proposed a protocol to guarantee the confidentiality of clients' local models through double-masking. The cloud server is required to provide proof of its aggregated results. Guo et al. [57] designed a high-efficient, drop-tolerant secure aggregation with verifiability. Their work is optimized for double-masking high-dimensional updates. Madi et al. [58] combined homomorphic encryption and verifiable computation in the federated aggregation, which support operations in the ciphertext domain and produce proofs along with the computed results. Xu et al. [59] proposed a non-interactive verifiable privacy-preserving aggregation using a dual-servers setting, which improves the efficiency of the system and is robust to client dropout.

2.2.3 Asynchronous Federated Learning

Mask-based PPFL is a highly efficient solution compared with HE and MPC solutions. However, strict synchronization is required and it degenerates the efficiency. When considering the clients in federated learning, the heterogeneity cannot be overlooked. Since federated learning, especially cross-device settings, is usually an open system, clients can differ in many aspects. First, the data owned by the clients may have different distributions, and the size of the local dataset may vary. Secondly, the clients' devices may have different computing power, power limit, and communication capability. Thirdly, the Internet connectivity of the clients' devices is different. Some

devices may use an unstable wireless connection. A synchronized aggregation protocol forces all parties in the system to wait for the "last comer" to submit its local model, so that the training on the clients' side is inefficient due to meaningless idling. Asynchronous federated learning can have higher utilization of computing power and communication bandwidth for devices, which increases the overall efficiency of the system. However, asynchronous federated learning cannot use the original federated aggregation algorithm, because the updates from clients are not based on the same version of a global model. To handle this problem, clients' models should always have tags to denote the version of the global model. The update of the global model can happen when the clients' updates come, by using the mixing algorithm [60], or when there is a certain number of updates from clients received, by using weighted aggregation with staleness function [61]. Both methods can modify federated learning into an asynchronous alternative. However, privacy preservation becomes a concern, because mask-based secure aggregation requires synchronization. The aggregation method of one-shot recovery [53] allows clients to communicate with the server without synchronization in the early rounds of one training epoch, but synchronization is still required when one-shot mask removal happens.

In asynchronous federated learning, when the server maintains a buffer and stores the coming updates, clients can communicate with the server at any given time, and when the server performs the aggregation and updates the global model, the server can orchestrate a temporary synchronization in the system. To preserve the privacy of any subset of local models, the server is only allowed to perform the aggregation when enough models are received. Therefore, threshold decryption/unmasking is adopted in the privacy-preserving buffed asynchronous federated aggregation. So et al. [62]

proposed a privacy-preserving asynchronous aggregation, which handles stragglers efficiently to increase efficiency and decrease the latency between the generation of two available global models. Their work adopts the staleness function [61] in the mask-based privacy-preserving aggregation. To remove the noise on the aggregated model, the linear homomorphism of the secure share is utilized. Chen et al. [63] proposed an asynchronous aggregation for vertical federated learning, which produces a global model using the local models trained on data with a different set of features. Chen et al. [64] proposed an asynchronous federated learning that allows clients with the non-IID dataset to contribute to the global model, which handles the device heterogeneity and data heterogeneity in the federated learning setting at the same time.

2.3 Comparison of Existing Works

A summary of existing works is given in Table 2.1 and Table 2.2, in which the underlying techniques and supported features are compared.

2.4 Summary

In this chapter, we have briefly introduced the cryptographic preliminaries, including public key encryption, homomorphic encryption, secret sharing, key agreement, homomorphic hash function, and commitment. Also, we have reviewed the existing schemes of privacy-preserving federated learning, verifiable federated learning, and asynchronous federated learning. However, it is still an open problem to achieve both the verifiability of model aggregation and the privacy preservation of local models efficiently.

Table 2.1: Summary of Existing Works (Techniques)

Paper	HE	DP	MPC	TEE	Masking	Other technique(s)
[40]	✓	×	×	×	×	–
[43]	✓	×	×	×	×	–
[35]	✓	✓	×	×	×	–
[36]	×	✓	×	×	×	Shuffling
[37]	×	✓	×	×	×	LDP
[41]	✓	×	×	×	×	–
[42]	✓	×	×	×	×	–
[44]	×	×	✓	×	×	–
[45]	×	×	✓	×	×	–
[35]	×	✓	✓	×	×	–
[46]	×	✓	✓	×	×	–
[48]	×	×	×	✓	×	–
[49]	×	×	×	✓	×	–
[50]	×	×	×	✓	×	–
[29]	×	×	×	×	✓	–
[53]	×	×	×	×	✓	–
[52]	✓	×	×	×	✓	–
[56]	×	×	×	×	✓	–
[57]	×	×	×	×	✓	–
[58]	✓	×	×	×	×	–
[59]	×	×	×	×	✓	–
[60]	×	×	×	×	×	–
[61]	×	×	×	×	×	–
[62]	×	×	×	×	✓	–
[59]	×	×	×	×	✓	–
[61]	×	×	×	×	✓	–
[63]	×	×	×	×	✓	–
[64]	×	×	✓	×	×	–

Table 2.2: Summary of Existing Works(Features)

Paper	Privacy-preserving	Verifiability	Asynchronous	Other feature(s)
[40]	✓	×	×	–
[43]	✓	×	×	–
[35]	✓	×	×	–
[36]	✓	×	×	–
[37]	✓	×	×	–
[41]	✓	×	×	Transfer learning
[42]	✓	×	✓	–
[44]	✓	×	×	–
[45]	✓	×	×	–
[35]	✓	×	×	–
[46]	✓	×	×	–
[48]	✓	×	×	–
[49]	✓	×	×	–
[50]	✓	×	×	–
[29]	✓	×	×	–
[53]	✓	×	×	–
[52]	✓	✓	×	–
[56]	✓	✓	×	–
[57]	✓	✓	×	–
[58]	✓	✓	×	–
[59]	✓	✓	×	Dual server
[60]	×	×	✓	–
[61]	×	×	✓	–
[62]	✓	✓	×	Dual server
[59]	✓	×	✓	–
[61]	✓	×	✓	–
[63]	✓	×	✓	–
[64]	✓	×	✓	–

Chapter 3

Privacy-Preserving Model Aggregation for Asynchronous Federated Learning

3.1 Introduction

With the fast development of the IT infrastructure, a vast amount of data becomes available, which makes machine learning-based solutions more feasible and less costly. As a new collaborative machine learning solution, federated learning breaks the data barriers across different clients to use more data for training without violating privacy regulations. The typical system topology is a client-server model, in which a group of clients collaborate with an aggregation server to constitute a "federation" for training a global model that benefits from data owned by different clients. The training process is interactive and requires clients and the server to communicate frequently in a synchronous manner. In each round of training, a client obtains the latest global model from the server, trains its local model using the local dataset, and sends the model updates (e.g., gradients, model parameters) to the server. The server generates the next version of the global model through a process called aggregation that takes the input as local model updates. In the training process, the raw data never departs

from its owner, which provides basic privacy protection to the clients participating in the training.

However, federated learning may leak a client's sensitive information even when only local updates are sent from clients. By utilizing the known attacks on the trained models, such as model reverse attacks and membership inference attacks, an adversary can invert a client's local model update to learn a large amount of information about its dataset [31]. Existing works use different methods to protect the privacy of local models. One approach is using difference privacy to hide the cleartext of local models before they are submitted, which has a strong mathematical guarantee on the maximum information that can expose for any sample used in the training. The noise added to local models cannot be removed in the aggregation and will be calculated into the global model. Adding noise to the final model can cause the degeneration of the accuracy of the global model. To protect the privacy of local models but introduce less noise to the aggregated model, other DP-based schemes [34–36] have been proposed and LDP [37] is the most widely accepted, in which each client adds DP noise independently and maintains a controllable level of the sum of noise. Cryptographic primitives also enable PPFL, which does not affect the result of aggregation. Homomorphic encryption has been adopted in PPFL to allow the server to perform the aggregation on the ciphertext of local models. By encrypting the local models, and only decrypting the global model for the server, the privacy of a certain client is protected. Additive homomorphic encryption, which is much more efficient than fully homomorphic encryption, can be used in the federated aggregation, because linear operations can meet all needs of the aggregation. For example, Paillier encryption [38], a homomorphic encryption scheme that supports unlimited numbers

of addition operations and multiplication by real numbers operations, can be used for weighted sum aggregation. Federated aggregation can also be redesigned as a privacy-preserving protocol by adopting MPC, in which the server and clients work collaboratively to enable aggregation without using the plaintext of local models. Both HE and MPC based works have much lower efficiency than the original federated learning, which makes them impractical in the real project.

In order to improve the efficiency, Bonawitz et al. [29] proposed Secure Aggregation (SecAgg) for federated learning. In secure aggregation, there is a contradiction between model privacy and model aggregation. To address this problem, SecAgg adds noise to local models and removes noise after aggregation. In the original federated learning, the server performs the linear operations on the cleartext of local models to calculate the global model. To achieve the same result when the cleartext of local models is not available, SecAgg requires 4 rounds of communication between the server and clients. The efficiency of this protocol is still lower than that of the original federated learning work. The main reason for this low efficiency is the synchronized workflow adopted, which causes all devices in the system to wait for the slowest one. Due to the data heterogeneity and device heterogeneity, different clients train and submit updates at a different frequency and some entity may wait for others, which cause the idling time on the server and clients to accumulate. So that the accumulated idling time becomes the largest part of the overhead. To build an efficient federated learning protocol, the asynchronous workflow should be adopted. The drawback of asynchronous federated learning is that the old updates can cause the degeneration of the model performance [53]. Therefore, a weighted asynchronous aggregation algorithm with staleness adaption [61] has been proposed to ensure that

the global model gets the most benefit from all updates coming at different times. Asynchronous secure federated learning has been proposed as a batch synchronized protocol, which looses the synchronization requirement and lets the clients have more flexibility when they submit updates.

Inspired by the asynchronous aggregation, we propose fully asynchronous secure federated learning, which allows clients to submit updates at any given time and protects the privacy of local models. The asynchronous workflow enables our protocol to make the most use of clients' computation power because the idling time caused by device heterogeneity no longer exists. This protocol also has better resilience of communication latency, which enables the training when clients have an unreliable network.

In this chapter, we propose a privacy-preserving asynchronous federated learning protocol(PPA-AFL) that enables secure aggregation of clients' local models and improves the efficiency of global model training. Specifically, the contributions of this paper are summarized in three folders:

- We address the conflict between local model leakage and local model aggregation by encrypting clients' local models with the Paillier encryption in asynchronous federated learning. The aggregation result remains the same, while the cleartext of clients' updates is never exposed.

- By adopting dual server setting and threshold secret sharing, the local model aggregation can only happen when a certain number of clients' updates arrives, which prevents the global model from leaking specific clients' information by decreasing the proportion of single client contributions in one aggregation.

- The proposed protocol is fully asynchronous from the view of clients. Clients

can decide when to join, and they do not need to be online until the aggregation. For clients, the overhead brought by the system is close to the original federated learning.

3.2 Problem Statement

We present the entities in the proposed protocols, the security model, and the goals of the proposed protocol.

3.2.1 Entities

The dual-server federated learning system consists of three entities: an encryption server, an aggregation server, and clients.

- The encryption server: The encryption server is a relatively powerful machine that has high computing ability, and a reliable network connection. The server does not have data for the training task. In the training, this server generates keys for homomorphic encryption and secret shares for threshold aggregation. This server can communicate with clients bidirectionally to allow clients to join the training at any time. The incentive of the server is to receive a commission from the duties of managing the cryptographic system.

- The aggregation server: The server is a relatively powerful machine that has high computing ability, reliable network connection, and large storage. The server does not have data for the training task. The incentive of the server is to produce a global model. This server maintains a buffer to store updates from clients and performs the aggregation on the ciphertext of local models. The aggregation result is sent to the other server for decryption.

- Clients: A client is a device with the local dataset. It is assumed that a client does not have high computing ability, reliable network connection, and large storage. A single client has limited data that is not diverse enough but the gathering of data from multiple clients can cover throughout the data distribution. To find benefits in a global model that has better generalization ability, clients are motivated to join federated training. Clients in this system can occasionally contribute to some rounds of aggregation.

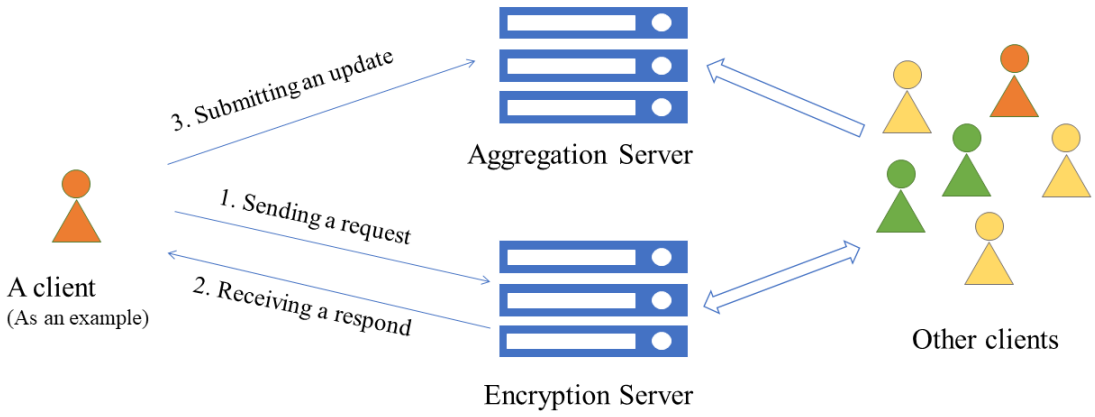

Figure 3.1: System Model for PPA-AFL

The system model is depicted in Fig. 3.1. In the original federated learning, there is only one server and multiple clients. In the proposed design, two servers are responsible for different parts of the protocol. The first server is called the aggregation server, which performs the aggregation of updates. The second server is called the encryption server, which distributes keys for encryption and decrypts the final result. The aggregation server maintains a buffer to save the updates from clients. The encryption server keeps the global model of different versions, and maintains the key pairs and secret shares for the next round of aggregation. The clients have their

dataset on the local storage, and train the local model based on the global model using local data.

In this system, communication does not happen between arbitrary two parties. Two servers can communicate. Clients and encryption servers can communicate bidirectional, while the communication between clients and aggregation is only from clients to the server.

Clients contribute to the global model by doing local training and submitting updates. The aggregation server performs the aggregation, which uses the local models' ciphertext and outputs the global model's ciphertext. The aggregation server should only expose the encrypted global model to the encryption server but not any individual encrypted update. The encryption server generates keys for homomorphic encryption and sends them to clients who claim they are about to do the local training. The encryption server also decrypts the ciphertext from the aggregation server to get the new global, which is sent to clients along with encryption keys. The encryption server should only decrypt the ciphertext of the global model when the global model is aggregated from a large enough number of clients' updates, which is implemented by using secret sharing.

3.2.2 Security Threats

There are three parties in our protocol, the encryption server, the aggregation server, and the clients. There are two threat models for our system:

- All entities in our system are assumed to be honest but curious.
- The aggregation server and the encryption server are honest but curious. Some clients are malicious, and the rest of the clients are honest but curious.

For every client in the system, it has a local dataset that it is not willing to share. The raw data in the dataset of other clients is not accessible in the federated learning, but the updates that contain the information of the local data can be analyzed to expose the sensitive information. Unprotected updates from clients can become a possible privacy leakage. Additionally, because a federated learning system is open and welcoming clients from the internet in practice, the communication channel may not be secure. The motivation of secure federated learning is to perform the federated learning, while protecting the clients' privacy at the same time.

There are three possible threats to the privacy of clients, when "all entities in our system are assumed to be honest but curious" assumption is applied:

- In our protocol, updates are sent from the clients to the aggregation server. The channel between a certain client and the aggregation server may not be secure, which means an attacker may access the updates of certain clients. The protocol needs to protect clients' privacy, even when their updates are known by an attacker.

- Aggregation happens on the aggregation server. The aggregation server performs a linear operation on the ciphertext of the clients' local model. The result is sent to the encryption server to be decrypted. The fewer updates are included in an aggregation, the more possibility of a certain client's privacy can leak. The minimum number of updates that must be reached to allow an aggregation start should be considered.

- The encryption server has keys to encrypt, decrypt, and perform the evaluation on the ciphertext. When a certain update is got by the encryption server, the

plaintext of that client's local model is exposed. The encryption server in our protocol should never receive a ciphertext of the model that only contains one's or a fewer clients' information.

In addition, when some clients in our system are malicious, possible threats to the privacy of clients are:

- Malicious clients can use a model in which every parameter is zero as its local model. When there are multiple malicious clients that work together, they can get the sum of other clients' local models. The extreme situation is that, in one aggregation, there is only one honest client and all the rest clients are malicious.
- Malicious clients can send requests to the encryption server at a high frequency to get more shares. When the number of shares surpasses the recovery threshold, it can determine when to start the next aggregation. With this advantage, it may exploit more information from honest clients.

3.2.3 Design Goals

The main goal is to achieve both privacy protection and efficient asynchronous model aggregation in federated learning. To achieve this goal, the following issues need to be addressed:

- Local model privacy: The locally trained models and gradients can still leak the sensitive information of the training sample. To protect the privacy of clients, both their local dataset and local models shall not be available in the form of plaintext for any other parties.

- Model aggregation conditions: The aggregated global model contains the information of local models. To reduce the information leakage of a specific sample, the global model must be aggregated from at least a certain number of local models, which means threshold aggregation should be adopted.

- The staleness of local models: In asynchronous federated learning, the aggregation server receives local models trained on different versions of global models. In order to ensure the validity of the model, the version of the local model needs to be tracked.

3.3 The Proposed PPA-AFL

3.3.1 Cryptographic Primitives

In the proposed work secret sharing and homomorphic encryption are adopted. These cryptographic primitives are defined in Chapter 2.1 Cryptographic Preliminaries.

To be specific, in this chapter, the algorithms are defined as follows:

- Shamir secret sharing: The Shamir secret sharing scheme includes three algorithms, as $SS = \{SS.setup(1^k), SS.share(n, t, m), SS.combine(t, \{[s_i]\}_{i \in P'})\}$.

- Paillier encryption: Paillier encryption is adopted for linear homomorphic encryption in this protocol. Paillier encryption supports addition and multiplication by real number operations on the ciphertext. It consists of the following five algorithms: $PE = \{PE.setup(1^k), PE.keygen(PE.param), PE.enc(m, pk), PE.eval(\{[c_i]\}, f), PE.dec(c, sk)\}$.

3.3.2 The Detailed PPA-AFL

There are three kinds of parties in our protocol, the encryption server, the aggregation server, and the clients. During the initialization of the system, the encryption server calls $SS.setup(1^k)$ and $PE.setup(1^k)$ to get parameters. Because our proposed protocol is asynchronous, the running process cannot be described in a serial fashion.

3.3.2.1 Encryption Server

This server maintains a global accessible value tag, which indicates the version of most variables and messages in this system. At the beginning of the protocol, the tag is set to $tag = 0$. During the running of protocol, when the current value tag is n, the new tag is assigned as $n + 1$ when the tag needs to be updated.

We assume there is a tag currently maintained by the server, and the server behaves following the description below.

- After the aggregation or when $tag = 0$

  The current value of tag is v.

  The server randomly generates a secret value s_v. The server calls the $\{[s_{v,i}]\}_{i\in[0,n)} \leftarrow SS.share(n_v, t_v, s_v)$, where n_v controls the number of shares and t_v controls how many updates are needed to allow the aggregation happens. n is a large enough integer that should always be greater than the number of updates in the system. t is selected by the encryption server to limit the behaviour of the aggregation server.

  The server calls $(pk_v, sk_v) \leftarrow PE.keygen()$.

- When receiving a request from a client i

The server responds to the clients a message with the following content: current tag v, public key pk_v, one secret share that has not been sent to others yet $s_{v,x}$, and latest global model M_v.

- During the aggregation

  When receiving the ciphertext of the new global model $c_{v,M}$ and shares $\{[s_{v,i}]\}_{i\in[0,t')}$ from the aggregation server. The server checks:

  - if $t' >= t$,
  - $s_v = SS.combine(t_v, \{[s_{v,i}]\}_{i\in[0,t')})$.

  When the check result is passed, the server calls $M_{v+1} \leftarrow PE.dec(c_{v,M}, sk_v)$ to get the new global model; otherwise, the aggregation fails and the *tag* and global model are unchanged. This information should be sent to every party in the system as a notice.

3.3.2.2 Clients

When the client i wants to join the train, it sends a request to the encryption server. In the response from the server, it gets the current tag v, public key pk_v, one secret share $s_{v,x}$, and the latest global model M_v. This client runs the local training to get $m_{v,i,count}$. *count* indicates this is the $count^{th}$ local model under the same v, this information may be omitted in the following parts. The client calls $c_{v,i,count} \leftarrow PE.enc(m_{v,i,count}, pk_v)$ to get the ciphertext of the local model. The client sends an update to the aggregation server with the following content: encrypted local model $c_{v,i,count}$, and a secret share $s_{v,x}$.

To join the training again, the client needs to send a request to the encryption

server again. Generally, there is no limit on how many times a client can create updates under the same *tag*.

3.3.2.3 Aggregation Server

The aggregation server maintains a buffer to save the updates from clients temporarily.

When the updates with the same $tag = v$ surpass the value of t_v, the server calls $c_{v,M} \leftarrow PE.eval(\{[c_{v,x}]\}, f)$ to get the ciphertext of the aggregation result. f is the aggregation algorithm, and the same one in the original federated learning can be adopted.

The shares in these updates are packaged as $\{[s_{v,x}]\}$. The subtext x is used for simplicity, while the local models and shares are still from clients with information like "client i" and "update times *count*".

The updates with the old *tag* whose related global model has been decrypted by the encryption server are discarded. For the new updates with different *tag*, the first one that reaches the number of t_v is processed as above. The message sent to the encryption server contains the corresponding *tag*, the ciphertext of global model $c_{v,M}$, and a set of shares.

3.4 Security Analysis

We discuss the security goals, namely, the privacy of local models, and threshold aggregation under two threat models of our system, "all parties are honest but curious" and "some clients are malicious". When the "all parties are honest but curious" assumption is applied, we prove that the privacy of the local model is protected and

the model aggregation cannot happen before the threshold is reached. A client who is ready to join the federated training firstly sends a request to the encryption server. The response contains an encryption key generated by the encryption server. A client performs local training and uses the encryption key to encrypt the local model. All encrypted local models from clients are sent to the aggregation server. The aggregation performs on the ciphertext of local models. The aggregate result is sent to the encryption server and decrypted. In this process, the plaintext of local models never departs from the local device. The ciphertext of local models is sent from clients to the aggregation server, which means the aggregation server and eavesdroppers on the channels can see the ciphertext. According to the Paillier assumption, without the decryption key, the probability of the adversary distinguishing a ciphertext and a random string from cipher space, is negligible. So that, neither the aggregation server nor eavesdroppers can extract any information from the ciphertext of local models. When the encryption server sends an encryption key to a client, one piece of secret share is also provided. These shares are sent to the aggregation server when clients submit their updates. When the aggregation server requests the encryption server to decrypt a ciphertext of the global model, these shares need to be provided. If the secret can be recovered from shares, the encryption server performs the decryption; Otherwise, the request is rejected. According to Shamir's secret sharing scheme, the secret cannot be reconstructed with the insufficient number of shares, which forces the aggregation server to perform the threshold aggregation honestly. When the "some clients are malicious" assumption is applied, malicious clients can send requests to the encryption server at a high frequency to get more shares. In our proposed protocol, a client can only get encryption keys and secret shares from the encryption server. To

request shares, a malicious client has to expose its identity to the encryption server. Maintaining a record of requests by the encryption server can detect this attack easily. So that, the abnormal client and related aggregation can be suspended.

3.5 Evaluation

To evaluate the performance of our proposed protocol, we implement it with java. In this section, the time complexity of different parties is analyzed, and the running time in practice is shown.

3.5.1 Complexity Analysis

For simplicity, some definitions are given here.

- Global models and local models are of size m.
- The number of updates with the same tag is u.
- "t out of n" secret shares are generated by the encryption server.

The following part discusses the communication cost and computation cost of our protocol for different parties.

- For clients:
  - Communication cost

    $O(m)$ for each update. The cost to send a request to the encryption server is $O(1)$. The cost to receive the response from the encryption server is $O(1)$. The cost to send the update to the aggregation server is $O(m)$.

  - Computation cost

    $O(m)$ for each update. The cost to encrypt the plaintext of the local model is $O(m)$.

- For the encryption server:

  - Communication cost

    $O(u)$ for the period between two aggregations. The cost to receive a request from clients is $O(u)$ and the cost to send a response to clients is $O(u)$.

    $O(m)$ for each aggregation. The cost to receive a ciphertext of a global model from the aggregation server is $O(m)$.

  - Computation cost

    $O(n)$ for the period between two aggregations. The cost to generate "t out of n" shares is $O(n)$.

    $O(u^2)$ or $O(m)$ for each aggregation, the greater one should be applied. The cost to check the recovery result from shares is $O(u^2)$, and the cost to decrypt the global model is $O(m)$.

- For the aggregation server:

  - Communication cost

    $O(mu)$ for the period between two aggregations. The cost to receive updates from clients is $O(mu)$.

    $O(m)$ for each aggregation. The cost to send a ciphertext of the global model to the encryption server is $O(m)$.

  - Computation cost

$O(mu)$ for each aggregation. The cost to perform the aggregation is $O(mu)$.

3.5.2 Running Time on The Prototype Implementation

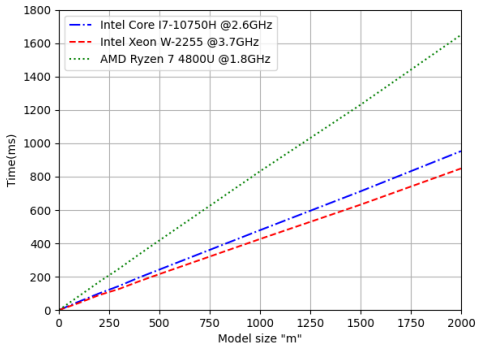

Figure 3.2: Time Consumption for Encrypting a Local Model by A Client

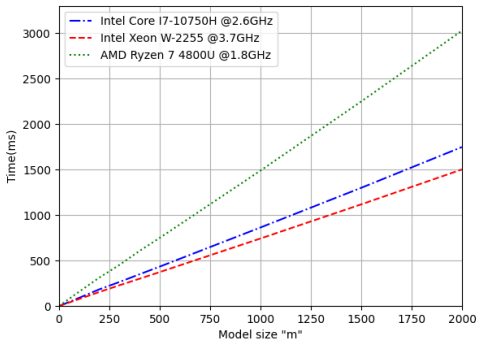

Figure 3.3: Time Consumption for Decrypting A Global Model by The Encryption Server

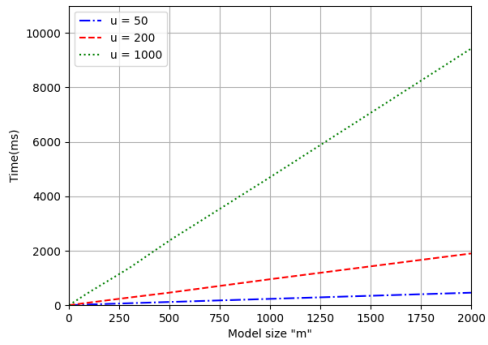

Figure 3.4: Time Consumption for Aggregation by The Aggregation Server (With Different "u")

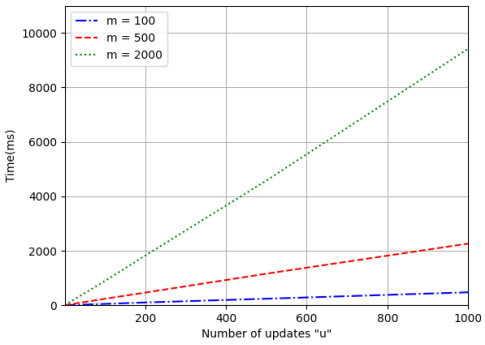

Figure 3.5: Time Consumption for Aggregation by The Aggregation Server (With Different "m")

We implement our protocol in java and get the running time. The computers we use have the CPU Intel Core i7-10750H, Intel Xeon W-2255, and AMD Ryzen 7 4800U.

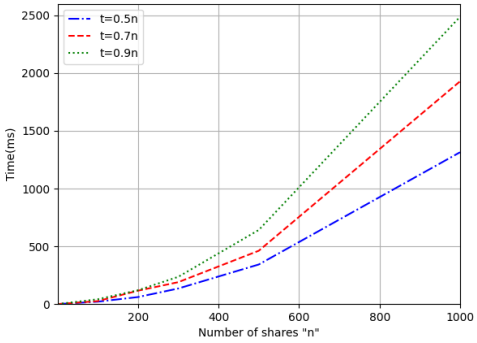

Figure 3.6: Time Consumption for Re-Generating Secret Shares by The Encryption Server

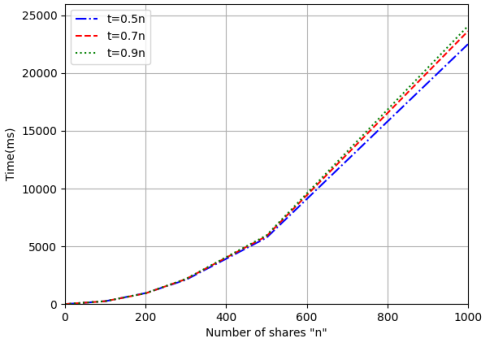

Figure 3.7: Time Consumption for Recovering Secret from Shares by The Encryption Server

The CPU frequency is locked at 2.60GHz, 3.70GHz, and 1.80GHz, respectively. The above-mentioned hardware is selected to simulate the setting of a relatively powerful personal PC, a common business server, and a slim laptop. The operating system is the latest version of Windows 11. All programs only use a single thread to run.

Because the proposed protocol is an asynchronous one, most operations can run in parallel. The efficiency of the system is determined by the slowest operation of each party. The result is shown in Fig 3.2-3.7. Following are the detailed explanations.

To evaluate the performance of the homomorphic encryption adopted in this protocol, we use three hardware settings mentioned above to run the encryption and decryption algorithm, the time consumption is shown in Fig 3.2 and Fig 3.3. From the result, the time consumption is linear to the model size m. When assuming the model size is 1000, which is a common value in practice, a slim laptop can perform the encryption in about 800ms and a business server can perform the decryption in about 750ms. This reflects that the cryptosystem adopted in our protocol can be implemented with high efficiency.

To evaluate the performance of the homomorphic aggregation, we use the computer with Intel Xeon W-2255 to run the homomorphic evaluation algorithm, the time consumption is shown in Fig 3.4 and Fig 3.5. From the result, the time consumption is linear to the model size m and the number of updates u. In practice, training a more complex model (the size m is bigger) often requires more updates to be submitted, which causes the time consumption to increase more than linearly when the model gets more complex. There should be a trade-off between the model size m and the number of updates u. Within the reasonable range, decreasing the number of updates u for aggregation each time can reduce the time consumption. The criterion of how many updates should be used in the aggregation is based on the specific model.

To evaluate the performance of the secret sharing scheme adopted in this protocol, we use the computer with Intel Xeon W-2255 to run the sharing and recovery algorithms, the time consumption is shown in Fig 3.6 and Fig 3.7. From the result, the time consumption for generating and recovering shares is quadratic to the number of shares n and recovery t. The selection of n is based on the maximum number of updates allowed in the aggregation, and the selection of t is based on the minimum number of updates allowed in the aggregation. In practice, a more frequent aggregation with fewer updates can increase the performance of the system.

3.6 Summary

We proposed a fully asynchronous secure federated learning protocol, which reduces the influence of device heterogeneity during the federated training. By using our proposed protocol, the system can make maximum use of clients' computing

power, which lets the aggregation happen more frequently when other factors remain unchanged. With asynchronous aggregation, the communication latency also has less impact on the system. Both these two advantages enable the protocol to perform well in practical use. The disadvantage of our protocol is that it requires two non-colluding servers. This setting makes our protocol cannot be implemented when there are only one party can serve as the server or the parties playing the role of servers have the motivation to collude.

Chapter 4

Privacy-Preserving Verifiable Buffered Asynchronous Federated Learning

4.1 Introduction

As a new machine learning paradigm, federated learning breaks the data barriers across different clients under privacy-protection constraints. The typical architecture is a client-server model, in which a group of clients collaborate with a server (also known as an aggregator) to constitute a "federation" for training a machine learning model that benefits from data owned by different clients. The training process includes multiple rounds of interactions between the clients and the server. In each round, a client obtains the global model from the server, trains its local model based on the local dataset and the global model, and sends the model updates (e.g., gradients, model parameters) to the server. The server updates the global model through a process called aggregation that takes the input as local model updates of clients and outputs the new global model. In the training process, the raw data never departs from its owner, which provides basic privacy protection to the clients participating in the training.

However, the local updates may leak clients' private data when they are shared with the server. Federated learning does not have any commitment to the honesty of the server. In particular, in cross-silo federated learning, the clients exploit a server offering "learning as a service" to build the learning architecture. It is hard to guarantee that the server is fully trusted. Even worse, the server may be compromised and fully controlled by a malicious attacker. By utilizing the known attacks on the trained models, such as model reverse attacks and membership inference attacks, an adversary can invert an individual model update of a target user to learn a large amount of information about its dataset. For this reason, Bonawitz et al. [29] have proposed Secure Aggregation (SecAgg) for federated learning. SecAgg is based on secret sharing that allows a client to share her random noise added to the model update with other clients participating in the training process and the clients contribute their received shares when they submit their model updates, so that the server can remove the noises on the aggregated global model. The security guarantee is the same as the standard secret sharing, so SecAgg is believed to be one of the best methods against model inversion and related inference attacks. Subsequently, many variants have been proposed. For example, Kadhe et al. [65] proposed a fast secure aggregation protocol based on fast Fourier transform that is efficient in communication and computation and robust to client dropouts. So et al. [66] designed a Byzantine-resilient secure aggregation framework to resist the model pollution of adversarial byzantine users, who may manipulate the global model by modifying their local updates or datasets.

However, SecAgg and its variants commonly depend on synchronous training, which means that the server has to wait for sufficient updates from clients at each

round. As a result, the convergence performance is significantly degraded. Asynchronous aggregation address this issue by enabling the server to aggregate the model updates at any time it receives, but SecAgg is not compatible with asynchronous aggregation unfortunately, because the server has to aggregate a sufficient number of local models to remove the random noises added by the clients for local update protection in each round. So et al. [62] proposed the first Buffered Asynchronous Secure Aggregation (BASecAgg) that extends SecAgg in a buffered asynchronous setting. The buffered asynchronous aggregation in federated learning, known as FedBuff, supports the server to store the local updates in a buffer once upon receiving and aggregate the updates to produce a global model when the buffer is full. Another approach to achieving secure asynchronous aggregation in federated learning is based on local differential privacy, in which each client independently adds noise to the local model update before uploading to the server. These noises, nevertheless, degrade the training accuracy. In addition, FedBuff can be extended to support FedAgg, but trusted execution environments, such as Intel software guard extensions (SGX), are required.

In SecAgg, the server is required to aggregate the local models from clients after the number of received model updates exceeds the threshold in each round; otherwise, the produced global model is invalid. Nevertheless, there are not methods in SecAgg to enforce the server to wait for enough local updates. The clients are not aware of the deviation of the global model brought by the unerased noises. In addition, it is hard to guarantee that the server honestly aggregates the local models. Computation errors are sometimes inevitable. A "lazy" server may randomly select the parameters of the global model instead of aggregating local models. This misconduct is baseless,

particularly in cross-silo federated learning. Therefore, to guarantee convergence performance, it is critical for the clients to have the capability that checks the correctness of the global model distributed by the server in each round. Xu et al. [56] proposed the first privacy-preserving and verifiable federated learning framework from SecAgg, in which the clients can verify the correctness of their aggregated results based on the generated proof of the server. Luo et al. [67] and Guo et al. [57] further improve communication and computation efficiency on model aggregation in federated learning. However, none of them can support the verification of asynchronous aggregation.

In this section, we address the issues of privacy leakage, model verifiability, and asynchronous aggregation in federated learning and propose PPVA-AFL, a novel secure and verifiable buffered asynchronous aggregation protocol for federated learning. PPVA-AFL enables clients to directly upload their local models with the predistribution of the secret shares of the noise and verify the received global model to ensure its correctness in asynchronous federated learning. To handle the unsynchronized model updates, a buffer is maintained by the server to store the clients' model updates, along with the tags that indicate the versions of the clients' updates. The synchronous aggregation should be tolerant to the aggregation of model updates with different versions. Each model update is protected by the pseudo-random noise, which is shared to other clients for noise removal in model aggregation. The unmasking process is designed as a one-shot recovery, which can allow the server to request all information needed to remove the noise in one round of communication with clients. Drop tolerance is also achieved as the secret share with Maximum Distance Separable (MDS) code is adopted to remove noise added by a dropped client. Furthermore, to guarantee the correctness of the global model, verifiability is achieved based on

the homomorphic hash function. Specifically, each client generates the homomorphic hash tag for a model update and sends the tag to the server, along with the model update. Then, the server can aggregate the homomorphic hash tags from the clients in the same way as model aggregation to generate the global model proof, which is used for the clients to verify the correctness of the local model. In short, the main contributions are summarized in three-fold.

- PPVA-AFL is proposed for secure buffered asynchronous aggregation in federated learning, which preserves the privacy of the local model updates of the clients submitted at any time and supports secure aggregation of local model updates stored in the buffer for global mode generation.
- PPVA-AFL addresses the challenge of verifiable secure asynchronous aggregation, which enables the client to check the correctness of the global model based on the proof of the server that is aggregated from the homomorphic hash tags from the clients.
- PPVA-AFL is proven to prevent information leakage from the local model updates, while enabling local model aggregation and global model verification. The computational and communication efficiency of PPVA-AFL is demonstrated with complexity analysis and extensive experiments. The convergence performance of the global model is also shown with respect to different network latencies and different staleness penalties.

4.2 Problem Statement

We present the entities in the proposed protocols, the security model, and the goals of the proposed scheme.

4.2.1 Entities

Similar to the original federated learning scheme, there are two entities in the proposed one, the server and the clients.

- The server: The server usually does not own the data, but helps clients complete the training of the global model in federated learning. Considering the number of clients and frequent data transmission in cross-silo federated learning, the server is required to have powerful computing power, reliable and high-bandwidth communication, and a stable power supply. There are two possible motivations for a server to participate in federated learning, as a beneficiary of the global model or hired by a group of clients.

- Clients: Clients are data contributors in this collaborative machine learning scheme, which can be individual data owners or a device with data. Considering hardware conditions, clients may not have the computing power to use a large amount of data for training, and communication and the power supply may also be limited. Considering the quality of data, the data of a single client is not diverse enough, and training with a large number of datasets of clients can usually make the model perform better. Out of optimism about the generalization ability of the global model, the client has the motivation to join federated learning.

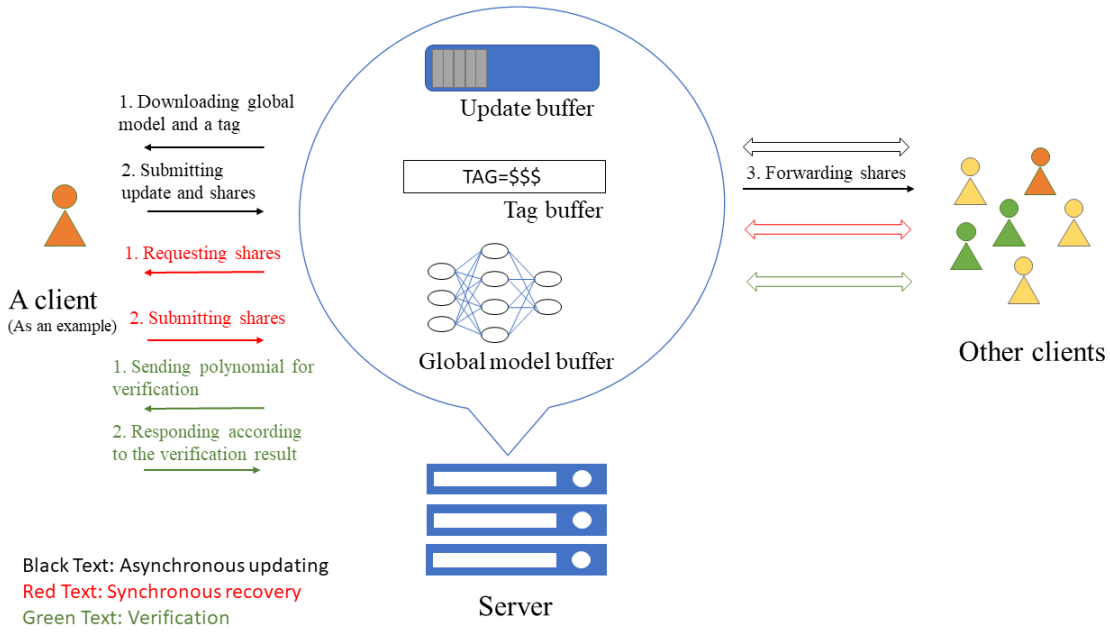

Figure 4.1: System Model for PPVA-AFL

The system model is depicted in Fig. 4.1. The proposed scheme is asynchronous for clients sending data to the server, but synchronization is still occasionally needed in order to aggregate the local model to get the global model. As shown in the illustration, there are three types of communication in the protocol, namely: asynchronous update uploading represented by black text; secret share collection in order to remove the noise during aggregation represented by red text; optional global model verification by clients represented by green text. The act of the client sending updates to the server is always asynchronous and happens at any time. The server temporarily stores updates received from clients in a buffer. The server can decide whether to initiate a local model aggregation based on the number of updates in the buffer or the time since the last update. Due to the adoption of the one-shot secret recovery method, the server only needs one round of communication with all online clients to

calculate the noise-free result when performing aggregation. After the aggregation is complete, the new model and other auxiliary information are made public. Clients can use the updated model for local training and can optionally perform verification on the previous global model.

4.2.2 Security Threats

The clients are honest-but-curious which means performing operations in full accordance with the protocol definition, but all data generated and passed through the entity can be viewed and analyzed. The server is honest-but-curious or "lazy". A "lazy" server does not intentionally attack against privacy, but tries to reduce its workload by acting dishonestly. For privacy-preserving and verifiable federated learning, the privacy of local models and the correctness of global models are two security goals.

There are three possible threats against the privacy of local models:

- Clients send the results of local training to the server for aggregation. The plaintext of this information can be used to exploit the privacy of clients, so it is necessary to be hidden.

- The server uses the local model to aggregate a global model. Any local model involved in the calculation contains the privacy of a specific client. The aggregation process should not expose any specific local updates.

- Clients add noise to their own updates and use secret sharing to keep the global model correct when they go offline. This behaviour should not lead to privacy breaches.

There are two possible threats against the correctness of global models:

- The server can be a "semi-lazy" server, which performs the model aggregation using only a subset of local models received.
- The server can be an "lazy" server, which uses a model filled with random values as the aggregation result.

4.2.3 Design Goals

The proposed scheme aims to address secure, verifiable, asynchronous aggregation simultaneously. To achieve this goal, the following issues need to be addressed:

- Local model privacy: For any clients participating in training, their local data and model should not be leaked to other entities.
- Verifiability of model aggregation: As the final output of federated learning, the global model should be correctly calculated from the local models used, even if the cryptographic primitives are adopted.
- The staleness of local models: Asynchronous updates make the local model based on different versions of the global model, and the fast convergence of the global model needs to be guaranteed to make the scheme practical.

4.3 The Proposed PPVA-AFL

4.3.1 Cryptographic Primitives

In the proposed work key agreement, symmetric encryption, secret sharing, and linear homomorphic hash are adopted. These cryptographic primitives are defined in

Chapter 2.1 Cryptographic Preliminaries.

To be specific, in this chapter, the algorithms are defined as follows:

- Key agreement: A two-party agreement scheme can be both used to create the key for symmetric encryption. This includes three algorithms, as $KA = \{KA.setup(1^k), KA.keyGen(KAparam), KA.agree(sk_i, pk_j)\}$.
- Symmetric encryption: The symmetric encryption includes three algorithms, as $SE = \{SE.keyGen(1^k), SE.enc(m, k), SE.dec(c, k)\}$.
- Secret sharing: The symmetric encryption includes three algorithms, as $SE = \{SS.setup(1^k), SS.share(P, t, m), SS.combine(t, \{[s_i]\}_{i\in P})\}$.
- Linear homomorphic hash: Linear homomorphic hash scheme concludes three algorithms, as $LHH = \{LHH.hGen(1^k, 1^d), LHH.hash(m), LHH.lEval(H, A)\}$. To be noted, this hash function works along with multi-dimensional data.

4.3.2 The Detailed PPVA-AFL

In this section, we give the detailed design of our proposed protocol. The setup shown in "AFL-Setup" only happens once at the beginning of the federated training. The local training shown in "LMod-Train" and the verification round shown in "GMod-Verif" is run on the clients' side in an asynchronous manner. The aggregation round shown in "GMod-Aggreg" is run on the server side, which needs some interaction with online clients to get shares for the secret recovery.

4.3.2.1 AFL-Setup

Definition 4.1 (Stochastic rounding function):

$$Q_c(x) = \begin{cases} \frac{\lfloor cx \rfloor}{c} & with\, the\, prob.\, 1 - (cx - \lfloor cx \rfloor) \\ \frac{\lfloor cx \rfloor + 1}{c} & with\, the\, prob.\, cx - \lfloor cx \rfloor \end{cases}$$

For each client $i \in P$:

- Client i calls $(sk_i, pk_i) \leftarrow KA.keyGen()$, and sends the public key pk_i to the server.
- After receiving public keys $\{pk_j\}_{j \in P}$ of other clients $j \in P$ from the server, client i calls $KA.agree(sk_i, pk_j)$ to agree a symmetric key $k_{i,j}$ between client i and client j.

Server S:

- The server calls $LHHparam \leftarrow LHH.hGen(1^k, 1^d)$, $SSparam \leftarrow SS.setup(1^k)$, and $KAparam \leftarrow KA.setup(1^k)$. The server generates a real number c and determines a rounding function $Q_c()$. The server broadcasts the global parameters as $(k, d, c, Q_c(), LHHparam, SSparam, KAparam)$.
- The server receives public keys $\{pk_j\}_{j \in P}$ from clients and broadcasts them.

4.3.2.2 LMod-Train

Definition 4.2 (Mapping function):

$$\phi(x) = \begin{cases} x & if\, x \geq 0 \\ q + x & if\, x < 0 \end{cases}$$

For each client $i \in P$:

- Client i downloads the global model M, which comes along with a timestamp, T_{Ver}. Client i sets the local timestamp as $T_i = T_{Ver}$ to keep a record of the version of the global model based.

- Client i performs the local training based on M, and the local gradient is Δ_i. Quantized gradient is calculated as $\bar{\Delta}_i =: \phi(c_l \cdot Q_{c_l}(\Delta_i))$.

- Client i generates a random value $z_i \in \mathbb{F}_q^d$. The masked update is calculated as $u_i = \bar{\Delta}_i + z_i$ and sent to the server's buffer along with tags T_i and R_i. T_i indicates the version of the global model based, and R_i indicates that the corresponding update is the R_i-th update based on the same global model version.

- The set of clients is defined as P, and $|P| = N$. U is the targeted number of surviving clients, D is the number of tolerated dropped users, and T is a arbitrary value less than U. Client i calls $SS.share(P, U-T, z_i)$ to partition z_i into $U-T$ shares as $\{[z_i]_1, [z_i]_2, ..., [z_i]_{U-T}\}$. In addition, T random values are generated as $\{[n_i]_{U-T+1}, [n_i]_{U-T+2}, ..., [n_i]_U\}$. These U values are encoded by using a (N, U) Maximum Distance Separable (MDS) code as:

$$[\tilde{z}_i]_j = ([z_i]_1, [z_i]_2, ..., [z_i]_{U-T}, [n_i]_{U-T+1}, [n_i]_{U-T+2}, ..., [n_i]_U) \cdot v_j,$$

  where v_j is the j-th column of a Vandermonde matrix $V \in \mathbb{F}_q^{U \times N}$, and $[\tilde{z}_i]_j$ is the secret share generated by client i for client j.

  Client i calls $LHH.hash(\bar{\Delta}_i)$ to get the hash value h_i.

  The message sent from client i to client j is as: $message_{i,j} = SE.enc(< [\tilde{z}_i]_j, h_i, count_i >, k_{i,j})$. This message is intended to be transferred by the server.

- When receiving the message from other clients, for example, the message from the client j, client i calls $SE.dec(message_{j,i})$ to get the plaintext $< [\tilde{z_j}]_i, h_i, R_i >$.

Server S:

- The server keeps a global model, with a timestamp T_{Ver}.
- The server transfers the messages to indicated clients.

4.3.2.3 GMod-Aggreg

A client can submit more than one update to the buffer. Therefore, for simplicity, u_k denotes the k-th update in the buffer and subtexts i, R_i are omitted. Other values, such as hash values and secret shares, related to this update are also with the subtext k.

For each client $i \in P$:

- After receiving the weighted aggregation function Ψ from the server, client i calculates the aggregated share of the mask, $z_i^{agg} \leftarrow \Psi[\{\tilde{z}_k\}_i]$, and sends z_i^{agg} to the server.

Server S:

- The server calculates the quantized staleness function as:

$$S_{c_g}() = c_g Q_{c_g}(s(\tau)).$$

- The server constructs the weighted aggregation function as:

$$\Psi(x_1, x_2, ..., x_k) = \sum_{i \in [1,k]} S_{c_g}(T_{Ver} - T_i) \cdot x_i.$$

- After receiving aggregated shares z_i^{agg} from clients, the server runs the MDS decoding to get $T-U$ shares as: $\{[Z]_1, [Z]_2, ..., [Z]_{T-U}\}$. The server calls $Z \leftarrow SS.combine(P, U-T, \{[Z]_1, [Z]_2, ..., [Z]_{T-U}\})$. Because of the linear homomorphism of the MDS coding and Shamir's secret sharing, the relationship $Z = \sum_{i\in[1,k]} \Psi(\{z_i\})$ holds.

  The quantized aggregation result is $\bar{\Delta}_{global} = \Psi(u_k) - Z$. The server performs the dequantization on the $\bar{\Delta}_{global}$ to get the Δ_{global}, as:

$$\Delta_{global} = \frac{1}{c_l c_g} \phi^{-1}(\bar{\Delta}_{global}).$$

  Δ_{global} can be used to update the global model. $\bar{\Delta}_{global}$ also needs to be accessible for clients for verification.

4.3.2.4 GMod-Verif

For each client $i \in P$:

- Client i checks if the $\bar{\Delta}_{global}$ maps to the Δ_{global}.
- Client i checks if $LHH.lEval(\bar{\Delta}_{global}) = LHH.lEval(\{h_k\}, \Psi)$; otherwise, verification fails due to the incorrect result from the server.

4.4 Security Analysis

We discuss the security goals, namely, the privacy of local models and correctness of global models under threat models of our system, "all parties are honest but curious".

When the "all parties are honest but curious" assumption is applied, we prove that the privacy of the local model is protected and the aggregation does not expose any specific local model cleartext. In the proposed scheme, clients add additive pseudo-random noise to local models before uploading them. This noise is not directly removed for aggregation, instead, the noise is removed from the noisy global model in one go after aggregation. Pseudo-random additive noise has a one-time-pad like property, which is generated before each update and used only once. Specific to the updates of a certain client, since its cleartext is added to the noise, its privacy is protected unless the attacker has effective methods of removing random noise, which does not exist yet. The sum of pseudo-random noise is still considered pseudo-random noise, which makes it impossible to gain any information about client privacy from the sum of noise. The correctness of the global model depends on the server honestly performing aggregation operations. In the proposed scheme, all clients calculate the corresponding hash value when generating the updates. The hash algorithm selected in this scheme supports additive homomorphism. After the aggregation, the client uses the aggregation function disclosed by the server to calculate the result taking input as hash values. By comparing the result with the hash as the input and the hash of the global model, a client can judge whether the aggregation has been executed correctly. If the server wants to forge the hash value so that the verification passes but the correct aggregation is not performed, it needs to use a hash collision, which is extremely difficult to find in the proven secure hash algorithm.

4.5 Evaluation

We build a prototype of our design with python and java. The model training and aggregation part is written in python, and the cryptographic primitives are implemented in java. In this section, all experiments assume a server, n clients and u updates in the buffer, where each update contains a local model in the form of a vector of size m. Because our protocol is asynchronous, the efficiency cannot be evaluated merely by the running time. The performance of clients and the server is analyzed with respect to different criteria. The performance analysis for the clients considers only one update because one client can submit multiple updates between two times of global updates, and different updates literally behave the same. For the view of the server, it repeatedly checks the updates in the buffer and requests the shares from online clients to update the global model. To evaluate the performance of the server, the time used to fill the buffer is ignored, and only the time used for calculating the new global model is considered. Convergence is necessarily achieved to output a useful model. An asynchronous workflow often hinders the convergence of a model. To show that our protocol is practical, we also do the experiment on the convergence with different sets of hyperparameters.

4.5.1 Complexity Analysis

4.5.1.1 Performance of Clients

- Computation cost

  $O(n)$ for the setup. It can be broken down to key generation cost $O(1)$ and key agreement cost $O(n)$.

$O(mu)$ or $O(n)$ for per update. It can be broken down to shares generation cost $O(n)$, MDS coding cost $O(1)$, hash computing cost $O(m)$, decryption for messages from other clients cost $O(u)$, shares aggregation cost $O(u)$, and verification cost $O(mu)$. n is the number of clients in the system, and u is the number of updates with the same tag at a certain time. When the number of clients is large and only a small percentage contributes to a certain version of the global, $O(n)$ should be considered. When the clients submit updates with relatively high frequency, especially in the situation that clients submit more than one update with the same tag, $O(mu)$ should be considered.

- Communication cost

  $O(n)$ for the setup. The cost to send the public key is $O(1)$ and the cost to receive other clients' public keys is $O(n)$.

  $O(n)$ for per update. The cost to send encrypted messages to other clients is $O(n)$ and the cost to receive encrypted messages from other clients is $O(n)$

  $O(u)$ for per aggregation happens. The cost to respond to the request from the server is $O(u)$ and the cost to receive the polynomial for verification from the server is $O(1)$.

  Based on the frequency of clients submitting their updates, n or u will be the most significant factor for the communication cost. When the submission frequency is low, n is the most significant factor; otherwise, u is considered more significant.

4.5.1.2 Performance of Server

- Computation cost

  $O(n)$ for the setup. It can be broken down to generating or selecting parameters for primitives $O(n)$.

  $O(u^2)$ for per aggregation happens. It can be broken down to calculating the polynomial for aggregation $O(u)$ and secret recovery $O(u^2)$.

  Secret recovery is the most expensive operation in the protocol. To reduce the overhead caused by $O(u^2)$ time complicity, the buffer size should be considered not too big to reduce the value of u.

- Communication cost

  $O(n)$ for the setup. The cost to receive and broadcast clients' public keys are $O(n)$ and $O(1)$ respectively.

  $O(n)$ for per update. The cost to receive the encrypted message from the client is $O(1)$, and the cost to send encrypted messages to other clients is $O(n)$.

  $O(n)$ for per aggregation happens. The cost to send requests for shares is $O(n)$. The cost to receive shares is $O(n)$. The cost to broadcast the polynomial for verification is $O(1)$.

4.5.2 Running Time on The Prototype Implementation

To show the overhead of our protocol in the real world, we use Java to implement the primitive we used in our protocol. By simulating all operations that happen following the protocol, we get the amount of time used by clients and the server for running a certain operation.

The computer we use for the simulation has the CPU Intel i7-10700k, and the CPU frequency is locked at 2.90GHz. The operating system is the latest version of Windows 11. All simulations only use a single process. We assume that $m = 1000$, which means the model has 1000 parameters. Through the analysis, the parameter selection of the secret sharing protocol has a great influence on the actual cost of the proposed work. We change the number of clients n and the secret share recovery threshold t to simulate the real-world situation (written as t out of n). All tests are run at least 1000 times and the average values are recorded as results. The results are shown in Table 4.1. The unit for the time is the millisecond(ms). The operations 1-5 are: 1. Hash value calculation by clients; 2. Global model verification by clients; 3. Secret share generation by clients; 4. Secret share aggregation by clients; 5. Secret recovery from shares by the server.

Table 4.1: Time Consumption for Certain Operations

	20 out of 30	50 out of 70	70 out of 100	700 out of 1000
Operation 1	13	13	13	14
Operation 2	331	674	998	8636
Operation 3	4	18	33	1799
Operation 4	1	1	1	3
Operation 5	24	126	256	22385

4.5.3 Convergence of The Global Model

Because our protocol works in an asynchronous manner, which means a lot of operations can run in parallel, time overhead is not the most concern when any of them does not have a high overhead. The significant concern for asynchronous aggregation is convergence. To test the convergence performance of our protocol, we implement our protocol in the mix of Python and Java. In this section, we first show

the convergence of the global model when there are different numbers of clients in the system. Then, we test our prototype, when different network latency is simulated. The hardware we use is the same as we used for the primitive performance test in the last section. The dataset we use is the MNIST.

- Convergence results w.r.t. different numbers of clients

  As shown in Fig 4.2 and Fig 4.3, we train the model with 20, 50, and 100 clients respectively. All of them can achieve convergence, and it is easy to achieve convergence with a less number of clients.

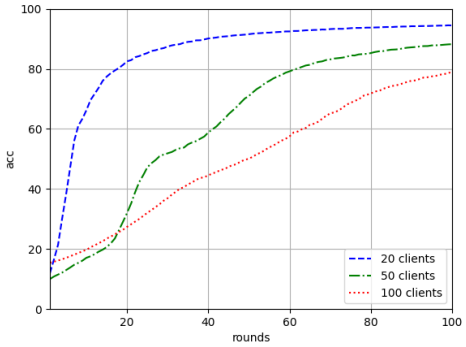

Figure 4.2: Accuracy w.r.t Different Numbers of Clients

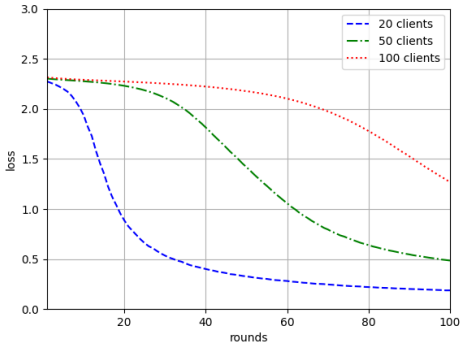

Figure 4.3: Loss w.r.t Different Numbers of Clients

- Convergence results w.r.t. different network latency

  To show how network latency affects convergence, the latency should be defined and simulated. In our protocol, the staleness function is used to set a penalty for out-of-date updates. We define that the mid latency is every update can have the possibility of 50% to be included in the possibly earliest aggregation. The

updates that miss the earliest round of aggregation have the possibility of 50% to be included in each coming round. Low latency and high latency are defined in the same manner, but with the possibility of 70% and 30% respectively.

As shown in Fig 4.4 and Fig 4.5, we train the model with low, mid, and high latency respectively. All of them can achieve convergence, and it is easy to achieve convergence with lower latency.

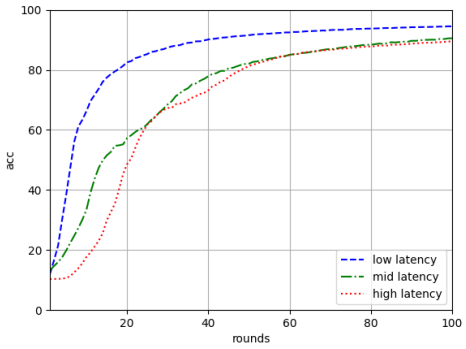

Figure 4.4: Accuracy w.r.t Different Network Latency

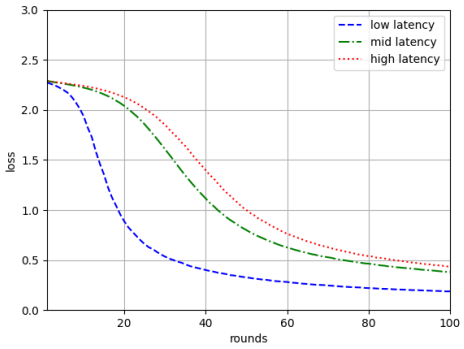

Figure 4.5: Loss w.r.t Different Network Latency

- Convergence results w.r.t. different staleness penalty

As shown in the last paragraph, higher latency is a disadvantage for the convergence, because out-of-date updates contribute less or even harm the global model performance. The method used to adjust this is the staleness penalty. In this section, we define the low, mid, and high penalty as 0.05, 0.1, and 0.2 respectively. For example, when mid-penalty is adopted, an update's weight in the weighted sum aggregation will be reduced by an additional 10% for each aggregation change it has missed.

We assume the latency is mid-latency and high-latency, and low, mid, and high penalties are adopted respectively. When mid-latency is adopted, the lower penalty has the best convergence result, as shown in Fig 4.6 and Fig 4.7. When high-latency is adopted, the high penalty has the best convergence result, as shown in Fig 4.8 and Fig 4.9. From the result, we can tell that within a reasonable range, higher penalties benefit the convergence when the latency is high because the higher penalties reduce the bad impact of the out-of-date updates.

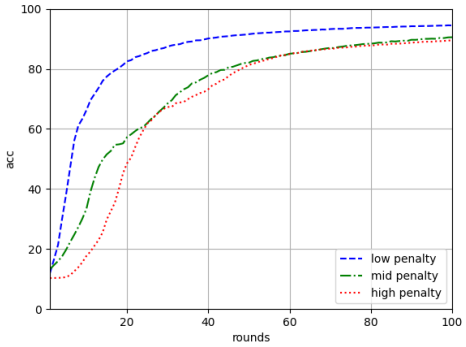

Figure 4.6: Accuracy w.r.t Different Latency Penalties (mid-latency)

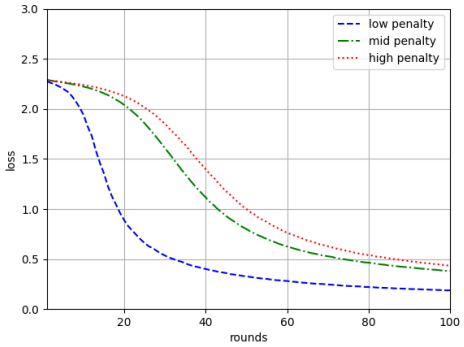

Figure 4.7: Loss w.r.t Different Latency Penalties (mid-latency)

4.6 Summary

We propose a secure and verifiable aggregation protocol for asynchronous federated learning. The privacy of the local model and the correctness of the global model can be strictly guaranteed simultaneously. Additionally, the proposed protocol enables asynchronous model updating and more flexible model aggregation. Thus, clients can adjust the frequency of uploading the local model according to their computing capability and network accessibility, and verify the correctness of the global

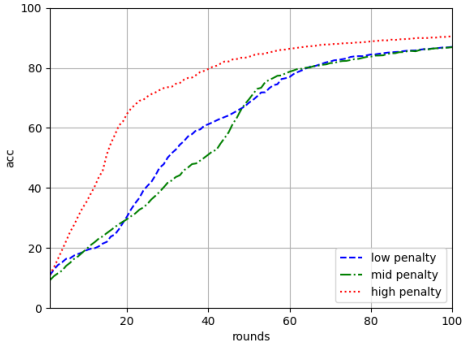

Figure 4.8: Accuracy w.r.t Different Latency Penalties (high-latency)

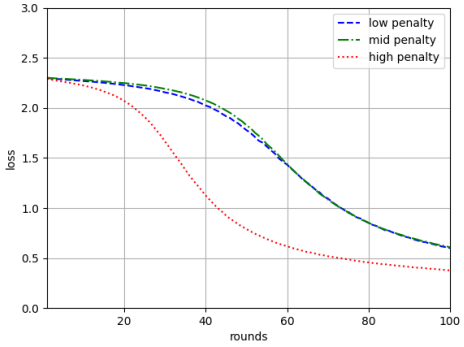

Figure 4.9: Loss w.r.t Different Network Latency (high-latency)

model using the time between two uploads. Aggregation can utilize more local models, including old models, to generate the global model without affecting the accuracy, which enhances the generalization ability of the global model and reduces the waste of computing power in the system. In short, we propose PPVA-AFL, which can perform secure and verifiable aggregation efficiently and asynchronously.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we have developed two novel schemes to solve the security and efficiency goals outlined for federated learning. In the PPA-AFL, the security goals are achieved through homomorphic encryption. The efficiency goals are achieved through an asynchronous workflow in the dual server setting. In the PPVA-AFL, the security goals are achieved through one-time-pad masking, and verifiable computation. The efficiency goals are achieved through asynchronous workflow, and one-shot secret recovery. We summarize the detailed contributions of the thesis below:

- To achieve privacy preservation for local models in the asynchronous aggregation, we propose PPA-AFL. Through this design, clients can participate in the training at any given time and the privacy of their local models is preserved. Homomorphic encryption is adopted to hide the cleartext of clients' updates and enable aggregation on the ciphertext. The key management problem is solved by the dual server setting in our design. Two servers' responsibilities are clearly spilt and the secure sharing scheme is adopted to ensure servers follow

the protocol. Asynchronous aggregation allows clients to decide when to join the federated training. The flexibility of participating provides convenience for clients, especially for cross-device federated learning.

- To support both the verifiability of model aggregation and the privacy preservation for local models in the asynchronous aggregation, we propose PPVA-AFL. Through this design, clients participate in the federated training asynchronously with their privacy preserved, additionally, the result of the model aggregation is verifiable to clients. Random masking is adopted to prevent sensitive information leakage from updates. The secure sharing scheme is adopted to allow the noise on the final model can be removed. Verifiable computation for the model aggregation is implemented by the homomorphic hash function. PPVA-AFL runs local model aggregation asynchronously and provides two expected security features at the same time, which encourages user participation in the practice.

5.2 Future Work

5.2.1 Blockchain-based Federated Learning

Traditionally, machine learning project requires centralized data collection and training in a single entity. Privacy concerns and the high overhead of raw data communications become an obstacle to large-scale use of machine learning [12]. Federated learning has been proposed to address the above two problems. However, the server in the federated learning is still a weak point of the system. Currently, there is no robust solution to prevent a malicious server [68]. To improve the system security of federated learning, blockchain is an attractive decentralized ledger technology. Blockchain

technology can be adopted to build serverless federated learning, which removes the impractical requirement of a trusted server. Training without the need for any central server makes federated earning more scalable.

5.2.2 Byzantine Resistant Federated Learning

The robustness has been a problem in multiparty computation. The byzantine problem in federated learning is slightly different. The task of federated learning is not just computation on distributed values but solving an optimization problem collaboratively. It is still an open problem how to identify a bad or malicious update in federated learning. Federated learning is designed to inherently benefit from distributed and heterogeneous databases owned by different parties. The contradiction is that bad updates are not similar to others but updates in federated learning inherently vary. The malicious client can follow the protocol and submit data in the expected form but with values not honestly gained from training [69]. This bad update cannot be inspected by any other parties in the federated learning with privacy preservation. Currently, there is no efficient method to filter out byzantine updates in federated learning. Byzantine resistance is a demanding feature in secure federated learning, because the availability of the final model is pivotal in federated learning.

5.2.3 Personalized Federated Learning

Federated learning enables multiple parties to train a single model collaboratively. It is assumed that all participants benefit from the final model. However, the performance of the final model on each client's data is different in practice. To address this problem, federated learning with personalization attracts attention. Personalized

federated learning produces slightly different models for different clients, which has better performance, especially for certain clients [70]. Personalized federated learning promises a better outcome than original federated learning.

5.2.4 Incentive Mechanism Design for Federated Learning

Clients in federated learning contribute to the global model with their heterogeneous data, and they lack the consistency in quantity and quality of their data [71]. In the original federated learning, all clients get the same reward, i.e., a global model, from participating in federated training, which is irrelevant to how much they contribute. This "unfair" reward discourages data owners. Federated learning with an incentive mechanism can compensate high contributors to maintain a healthy environment that allows the project continuously produce usable models. Designing incentive mechanism need to consider many factors, such as data set size, data accuracy, and data distribution [72].

Bibliography

[1] S. Sharma, M. Bhatt, and P. Sharma, "Face recognition system using machine learning algorithm," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 2020, pp. 1162–1168.

[2] M. Borg, C. Englund, K. Wnuk, B. Duran, C. Levandowski, S. Gao, Y. Tan, H. Kaijser, H. Lönn, and J. Törnqvist, "Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry," *arXiv preprint arXiv:1812.05389*, 2018.

[3] J. G. Richens, C. M. Lee, and S. Johri, "Improving the accuracy of medical diagnosis with causal machine learning," *Nature communications*, vol. 11, no. 1, pp. 1–9, 2020.

[4] I. Sadgali, N. Sael, and F. Benabbou, "Performance of machine learning techniques in the detection of financial frauds," *Procedia computer science*, vol. 148, pp. 45–54, 2019.

[5] Y. Roh, G. Heo, and S. E. Whang, "A survey on data collection for machine learning: a big data-ai integration perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 4, pp. 1328–1347, 2019.

[6] M. Goddard, "The eu general data protection regulation (gdpr): European regulation that has a global impact," *International Journal of Market Research*, vol. 59, no. 6, pp. 703–705, 2017.

[7] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[8] "Federated learning on google cloud," https://cloud.google.com/architecture/federated-learning-google-cloud, (Accessed on 11/15/2022).

[9] "Ibm federated learning," https://www.ibm.com/docs/en/cloud-paks/cp-data/3.5.0?topic=models-federated-learning-tech-preview, (Accessed on 11/15/2022).

[10] N. Rieke, J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein *et al.*, "The future of digital health with federated learning," *NPJ digital medicine*, vol. 3, no. 1, pp. 1–7, 2020.

[11] S. P. Ramu, P. Boopalan, Q.-V. Pham, P. K. R. Maddikunta, T. Huynh-The, M. Alazab, T. T. Nguyen, and T. R. Gadekallu, "Federated learning enabled digital twins for smart cities: Concepts, recent advances, and future directions," *Sustainable Cities and Society*, vol. 79, p. 103663, 2022.

[12] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

[13] S. Zhao, R. Bharati, C. Borcea, and Y. Chen, "Privacy-aware federated learning for page recommendation," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 1071–1080.

[14] G. Callebaut, G. Leenders, J. Van Mulders, G. Ottoy, L. De Strycker, and L. Van der Perre, "The art of designing remote iot devices—technologies and strategies for a long battery life," *Sensors*, vol. 21, no. 3, p. 913, 2021.

[15] J. P. Albrecht, "How the gdpr will change the world," *Eur. Data Prot. L. Rev.*, vol. 2, p. 287, 2016.

[16] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," *arXiv preprint arXiv:1812.02903*, 2018.

[17] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2938–2948.

[18] Y. Zhang, X. Yuan, J. Li, J. Lou, L. Chen, and N.-F. Tzeng, "Reverse attack: Black-box attacks on collaborative recommendation," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 51–68.

[19] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 3–18.

[20] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 739–753.

[21] G. Aridor, Y.-K. Che, and T. Salz, *The economic consequences of data privacy regulation: Empirical evidence from gdpr*. National Bureau of Economic Research Cambridge, MA, USA, 2020.

[22] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019-2019 IEEE international conference on communications (ICC)*. IEEE, 2019, pp. 1–7.

[23] O. Shahid, S. Pouriyeh, R. M. Parizi, Q. Z. Sheng, G. Srivastava, and L. Zhao, "Communication efficiency in federated learning: Achievements and challenges," *arXiv preprint arXiv:2107.10996*, 2021.

[24] W. Shi, S. Zhou, and Z. Niu, "Device scheduling with fast convergence for wireless federated learning," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[25] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," *Communications of the ACM*, vol. 59, no. 2, pp. 103–112, 2016.

[26] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, "Geppetto: Versatile verifiable computation," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 253–270.

[27] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[28] M. Chen, B. Mao, and T. Ma, "Fedsa: A staleness-aware asynchronous federated learning algorithm with non-iid data," *Future Generation Computer Systems*, vol. 120, pp. 1–12, 2021.

[29] 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*, 2017, pp. 1175–1191.

[30] K. Mandal and G. Gong, "Privfl: Practical privacy-preserving federated regressions on high-dimensional data over mobile networks," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2019, pp. 57–68.

[31] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 1–15.

[32] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.

[33] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*,

vol. 15, pp. 3454–3469, 2020.

[34] O. Choudhury, A. Gkoulalas-Divanis, T. Salonidis, I. Sylla, Y. Park, G. Hsu, and A. Das, "Differential privacy-enabled federated learning for sensitive health data," *arXiv preprint arXiv:1910.02578*, 2019.

[35] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM workshop on artificial intelligence and security*, 2019, pp. 1–11.

[36] A. Girgis, D. Data, S. Diggavi, P. Kairouz, and A. T. Suresh, "Shuffled model of differential privacy in federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 2521–2529.

[37] S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy, and W. Wei, "Ldp-fed: Federated learning with local differential privacy," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, 2020, pp. 61–66.

[38] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (Csur)*, vol. 51, no. 4, pp. 1–35, 2018.

[39] W. Du and M. J. Atallah, "Secure multi-party computation problems and their applications: a review and open problems," in *Proceedings of the 2001 workshop on New security paradigms*, 2001, pp. 13–22.

[40] 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," *arXiv preprint arXiv:1711.10677*, 2017.

[41] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 70–82, 2020.

[42] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[43] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "{BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning," in *2020 USENIX annual technical conference (USENIX ATC 20)*, 2020, pp. 493–506.

[44] H. Zhu, Z. Li, M. Cheah, and R. S. M. Goh, "Privacy-preserving weighted federated learning within oracle-aided mpc framework," *arXiv preprint arXiv:2003.07630*, 2020.

[45] R. Kanagavelu, Z. Li, J. Samsudin, Y. Yang, F. Yang, R. S. M. Goh, M. Cheah, P. Wiwatphonthana, K. Akkarajitsakul, and S. Wang, "Two-phase multi-party computation enabled privacy-preserving federated learning," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 410–419.

[46] D. Byrd and A. Polychroniadou, "Differentially private secure multi-party computation for federated learning in financial applications," in *Proceedings of the First ACM International Conference on AI in Finance*, 2020, pp. 1–9.

[47] 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. IEEE, 2015, pp. 57–64.

[48] A. Mondal, Y. More, R. H. Rooparaghunath, and D. Gupta, "Poster: Flatee: Federated learning across trusted execution environments," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2021, pp. 707–709.

[49] Y. Chen, F. Luo, T. Li, T. Xiang, Z. Liu, and J. Li, "A training-integrity privacy-preserving federated learning scheme with trusted execution environment," *Information Sciences*, vol. 522, pp. 69–79, 2020.

[50] X. Zhang, F. Li, Z. Zhang, Q. Li, C. Wang, and J. Wu, "Enabling execution assurance of federated learning at untrusted participants," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1877–1886.

[51] 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*, 2021, pp. 94–108.

[52] Z. Liu, S. Chen, J. Ye, J. Fan, H. Li, and X. Li, "Efficient secure aggregation based on shprg for federated learning," *arXiv preprint arXiv:2111.12321*, 2021.

[53] C.-S. Yang, J. So, C. He, S. Li, Q. Yu, and S. Avestimehr, "Lightsecagg: Rethinking secure aggregation in federated learning," *arXiv preprint arXiv:2109.14236*, 2021.

[54] Z. Peng, J. Xu, X. Chu, S. Gao, Y. Yao, R. Gu, and Y. Tang, "Vfchain: Enabling verifiable and auditable federated learning via blockchain systems," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 173–186, 2021.

[55] J. C. Garcia-Escartin, V. Gimeno, and J. J. Moyano-Fernández, "Quantum collision finding for homomorphic hash functions," *arXiv preprint arXiv:2108.00100*, 2021.

[56] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "Verifynet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2019.

[57] X. Guo, Z. Liu, J. Li, J. Gao, B. Hou, C. Dong, and T. Baker, "V eri fl: Communication-efficient and fast verifiable aggregation for federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1736–1751, 2020.

[58] A. Madi, O. Stan, A. Mayoue, A. Grivet-Sébert, C. Gouy-Pailler, and R. Sirdey, "A secure federated learning framework using homomorphic encryption and verifiable computing," in *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*. IEEE, 2021, pp. 1–8.

[59] Y. Xu, C. Peng, W. Tan, Y. Tian, M. Ma, and K. Niu, "Non-interactive verifiable privacy-preserving federated learning," *Future Generation Computer Systems*, vol. 128, pp. 365–380, 2022.

[60] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[61] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 3581–3607.

[62] J. So, R. E. Ali, B. Güler, and A. S. Avestimehr, "Secure aggregation for buffered asynchronous federated learning," *arXiv preprint arXiv:2110.02177*, 2021.

[63] T. Chen, X. Jin, Y. Sun, and W. Yin, "Vafl: a method of vertical asynchronous federated learning," *arXiv preprint arXiv:2007.06081*, 2020.

[64] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-iid data," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 15–24.

[65] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramchandran, "Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning," *arXiv preprint arXiv:2009.11248*, 2020.

[66] J. So, B. Güler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2168–2181, 2020.

[67] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-effective federated learning design," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[68] M. Grama, M. Musat, L. Muñoz-González, J. Passerat-Palmbach, D. Rueckert, and A. Alansary, "Robust aggregation for adaptive privacy preserving federated learning in healthcare," *arXiv preprint arXiv:2009.08294*, 2020.

[69] Y. Mao, X. Yuan, X. Zhao, and S. Zhong, "Romoa: Robust model aggregation for the resistance of federated learning to model poisoning attacks," in *European Symposium on Research in Computer Security*. Springer, 2021, pp. 476–496.

[70] V. Kulkarni, M. Kulkarni, and A. Pant, "Survey of personalization techniques for federated learning," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. IEEE, 2020, pp. 794–797.

[71] Y. Zhan, J. Zhang, Z. Hong, L. Wu, P. Li, and S. Guo, "A survey of incentive mechanism design for federated learning," *IEEE Transactions on Emerging Topics in Computing*, 2021.

[72] Y. Zhan, P. Li, Z. Qu, D. Zeng, and S. Guo, "A learning-based incentive mechanism for federated learning," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6360–6368, 2020.