Privacy-Preserving Byzantine-Robust Federated Learning via Blockchain Systems

Yinbin Miao, *Member, IEEE*, Ziteng Liu, Hongwei Li, *Senior Member, IEEE*, Kim-Kwang Raymond Choo, *Senior Member, IEEE*, and Robert H. Deng, *Fellow, IEEE*

***Abstract*— Federated learning enables clients to train a machine learning model jointly without sharing their local data. However, due to the centrality of federated learning framework and the untrustworthiness of clients, traditional federated learning solutions are vulnerable to poisoning attacks from malicious clients and servers. In this paper, we aim to mitigate the impact of the central server and malicious clients by designing a Privacy-preserving Byzantine-robust Federated Learning (PBFL) scheme based on blockchain. Specifically, we use cosine similarity to judge the malicious gradients uploaded by malicious clients. Then, we adopt fully homomorphic encryption to provide secure aggregation. Finally, we use blockchain system to facilitate transparent processes and implementation of regulations. Our formal analysis proves that our scheme achieves convergence and provides privacy protection. Our extensive experiments on different datasets demonstrate that our scheme is robust and efficient. Even if the root dataset is small, our scheme can achieve the same efficiency as FedSGD.**

***Index Terms*— Federated learning, poisoning attacks, fully homomorphic encryption, blockchain.**

I. Introduction

MACHINE Learning (ML) [1], [2] has been widely applied in various areas such as image recognition and medical diagnosis. With leveraging ML, valuable knowledge can be extracted from massive data to help users make informed decisions. However, training with a single data source results in an insufficient amount of data. To solve this problem, distributed machine learning [3], [4] is widely studied. Compared with traditional ML methods, distributed machine learning realizes the training of larger models on massive data, but is still vulnerable to security and privacy leakages. As an emerging paradigm of distributed machine learning, Federated Learning (FL) [5], [6] addresses learning tasks through an association of clients without exposing their original data. For relieving security concerns, privacy-preserving FL [22], [23] has been extensively studied among academic and industrial fields. One of the most common privacy-preserving federated learning schemes is to use the encryption schemes with homomorphism (e.g., Paillier) [20], [21] to encrypt the local gradients, which can be difficult to compute and transmit. Since Paillier can only encrypt integers and support encrypting individual data, thus the gradients need to be quantified first and encrypted one by one, which greatly increases the computation and communication overheads. Besides, the existing privacy-preserving FL solutions [25], [31], [32] still have another two issues to be solved.

The first issue is that privacy-preserving FL schemes are still subject to poisoning attacks [8], [9], [15]. For instance, a trained model may make incorrect predictions for a particular class while maintaining the accuracy of predictions for other classes (targeted attacks) [10]–[12]. Or the global model indiscriminately makes incorrect predictions for a large number of classes (untargeted attacks) [7]. To resist poisoning attacks, various aggregation rules such as Krum [14] and Trim-mean [13] have been proposed. Although these rules can remove malicious gradients through analyzing differences in the distribution of gradients in Euclidean space or ranking the magnitudes of gradients in each dimension, these rules are not always robust. For example, Krum and Trim-mean have been shown to be vulnerable to poisoning attacks [7]. Thus, how to provide a robust aggregation rule is the first issue to be solved.

The second issue is that privacy-preserving FL schemes suffer from malicious aggregation of the server and single point of failure threat. To solve this problem, Secure Multi-party Computation (SMC) [28], [36] is an alternative solution. However, the interaction processes of SMC incur heavy communication burdens on clients. As another way to address the server's malicious behaviors and single point of failure problem, blockchain becomes an increasingly promising solution [17], [18], [41]. However, the existing

Manuscript received 15 January 2022; revised 19 June 2022; accepted 25 July 2022. Date of publication 3 August 2022; date of current version 12 August 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 62072361 and Grant 62125205, in part by the Key Research and Development Program of Shaanxi under Grant 2022GY-019, in part by the Fundamental Research Funds for the Central Universities under Grant JB211505, and in part by the Henan Key Laboratory of Network Cryptography Technology and the State Key Laboratory of Mathematical Engineering and Advanced Computing under Grant LNCT2020-A06. The work of Kim-Kwang Raymond Choo was supported by the Cloud Technology Endowed Professorship. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Mark Manulis. *(Corresponding author: Ziteng Liu.)*

Yinbin Miao and Ziteng Liu are with the School of Cyber Engineering, Xidian University, Xi'an 710071, China, and also with the Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province, Hangzhou 310027, China (e-mail: ybmiao@xidian.edu.cn; liuziteng0330@163.com).

Hongwei Li is with the Department of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610051, China (e-mail: hongweili@uestc.edu.cn).

Kim-Kwang Raymond Choo is with the Department of Information Systems and Cyber Security, The University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: raymond.choo@fulbrightmail.org).

Robert H. Deng is with the School of Information Systems, Singapore Management University, Singapore 178902 (e-mail: robertdeng@smu.edu.sg).

Digital Object Identifier 10.1109/TIFS.2022.3196274

blockchain-based FL solutions still incur high computation overheads on blockchain, which is another challenging issue to be solved.

Recent works have provided us with some possible solutions to these issues. For example, Cao *et al.* proposed FLTrust [16], a federated learning framework against poisoning attacks, which uses a trusted root dataset and cosine similarity as criterias for judging malicious gradients. However, effective privacy protection is not considered in FLTrust. Therefore, Dong *et al.* proposed FLOD based on FLTrust, which also collects a root dataset [39]. The difference is that FLOD calculates the Hamming distance between the gradients and uses secret shares to provide privacy protection during the aggregation. However, FLOD cannot be deployed in distributed systems (e.g., blockchain network) to solve the single point of failure problem because the plaintext secret shares can be reconstructed by collusion servers or nodes. Similarly, Liu *et al.* proposed a Privacy-Enhanced Federated Learning (PEFL) framework to resist poisoning attacks [38]. However, the correlation between the gradients is revealed in PEFL, which may lead to inferences about the clients' information. And the single point of failure problem still exists in PEFL.

As we can see, previous FL solutions still cannot simultaneously resist poisoning attacks of clients and avoid malicious behaviors of the server while providing high security and efficiency, which do not scale well in practice. To solve these two issues, inspired by previous work [16], [38], we propose a Privacy-preserving Byzantine-robust Federated Learning (PBFL) scheme based on blockchain. Specifically, we adopt cosine similarity as a method to punish malicious clients. Different from previous schemes using cosine similarity mechanism, our scheme requires the server to collect a small and clean root dataset and maintain a model for the dataset. The server takes local updates and server model updates into consideration before determining the malicious gradients, which relies on a trusted root rather than only local updates as in the previous work. This is consistent with FLTrust [16]. Furthermore, we use homomorphic encryption as the underlying technology to add a privacy protection mechanism based on it. Besides, we use blockchain to facilitate transparent processes. The main contributions of our work are shown as follows:

- We provide a privacy-preserving training mechanism by using Fully Homomorphic Encryption (FHE) scheme CKKS [24], which not only greatly reduces the computation and communication overheads but also prevents attackers from snooping the local data of clients.
- We provide a trusted global model by removing the malicious gradients via cosine similarity, which resists the poisoning attacks.
- We use blockchain to facilitate transparent processes and enforcement of regulations. The servers perform off-chain calculations and upload the results to blockchain, which achieves both efficiency and credibility [19], [27].
- We demonstrate extensive experiments using two well-known datasets and compare our scheme with previous state-of-the-art schemes. The results show that our scheme performs well in terms of robustness and efficiency.

The rest of this paper is organized as follows: section II introduces the work in related fields of our paper. Section III introduces the preliminary knowledge. Section IV introduces the system model, design goals, and so on. The specific details of our scheme are described in section V. Section VI and Section VII give the security and performance analysis respectively, and finally we draw a conclusion in Section VIII.

II. Related Work

As our work progresses in several challenges of existing FL solutions, we focus on related work in the following two sub-topics of FL.

A. Federated Learning Against Poisoning Attacks

As the typical aggregation schemes in FL, FedAvg, and FedSGD [6] are widely used. However, these two schemes are vulnerable to poisoning attacks, which leads to the untrustworthiness and inaccuracy of the global model. To solve this issue, many new aggregation rules [13], [14], [22], [26], [38] have been proposed.

Blanchard *et al.* proposed Krum [14], a Byzantine-resilient algorithm for distributed stochastic gradient descent. Krum [14] selected one of n models in each iteration, which is most likely to be benign according to the Euclidean distance between the gradients. Similarly, Yin *et al.* proposed Trim-mean [13], in which the central server first sorted the model parameters and removed the larger and smaller parts of them, then calculated the average of the remaining values as the global model. Besides, Fung *et al.* [8] proposed FoolsGold, which calculates the cosine similarity between the historical gradients of the clients and resets the weights of clients to mitigate sybils in federated learning. However, the above schemes directly analyze and calculate the gradients in plaintext, which leads to privacy leakage. To solve this problem, Truex *et al.* [22] proposed a FL system, which uses Differential Privacy (DP) and secure multi-party computation to provide data privacy protection. The scheme [22] not only prevents inference attacks on sensitive data during the training but also provides high-precision models. However, the scheme [22] is vulnerable to malicious gradients, this is because encryption makes it difficult to calculate the similarity between local gradients. To solve this problem, Liu *et al.* proposed a Privacy-Enhanced Federated Learning (PEFL) framework [38]. PEFL adopts homomorphic encryption as the underlying technology, and uses the effective gradient data extraction of logarithmic function to punish malicious clients. However, PEFL cannot avoid malicious behaviors of the servers. In addition, the intermediate parameters in the training can be obtained by the servers, which causes inference attacks over clients.

B. Blockchain-Based Federated Learning

Traditional FL schemes heavily rely on the participation of the server, which may result in single point of failure and

TABLE I

A COMPARATIVE SUMMARY BETWEEN OUR SCHEME AND PREVIOUS SCHEMES

Schemes	Fun_1	Fun_2	Fun_3	Fun_4
Krum [14]	YES	✗	Central server	YES
Trim-Mean [13]	YES	✗	Central server	YES
FoolsGold [8]	YES	✗	Central server	YES
Scheme [22]	NO	DP	Central server	NO
PEFL [38]	YES	Paillier	Central server	NO
BAFFLE [18]	NO	✗	Bolckchain	NO
BlockFL [37]	NO	✗	Blockchain	NO
BFLC [33]	YES	✗	Blockchain	YES
PBFL	YES	CKKS	Blockchain	YES

Notes: Fun_1: Whether resisting poisoning attacks or not. Fun_2: Privacy-preserving mechanism. Fun_3: Central server or blockchain-based computation. Fun_4: Whether achieving lightweight computation or not.

malicious behaviors of the server. As an emerging technology, blockchain has been widely studied due to its decentralization and reliability. Many solutions [18], [33], [37] have been proposed by replacing the server with blockchain to eliminate the threat of single point of failure and the server's malicious behaviors.

Ramanan *et al.* [18] introduced a blockchain-based FL solution named BAFFLE, which uses Smart Contract (SC) to aggregate local models. Compared with traditional methods, BAFFLE avoids single point of failure and achieves lower gas costs for FL on blockchain. Similarly, Kim *et al.* [37] proposed BlockFL, in which the local model updates are exchanged and verified via a smart contract pre-deployed in blockchain. BlockFL not only overcomes the single point of failure problem, but also promotes the integration of more equipments and a larger number of training samples by providing incentives proportional to the size of training samples. However, the above solutions aggregate the local models via smart contract, which incur heavy computation and communication burdens on the nodes in blockchain network. In addition, these solutions cannot distinguish the malicious gradients. To solve these problems, Li *et al.* [33] proposed a Blockchain-based Federated Learning framework with Committee consensus (BFLC), which effectively reduces the amount of consensus computing and prevents malicious attacks. However, the selection criteria of the committee is a problem that need to be addressed.

Finally, TABLE I gives a comparison of the PBFL system with the above-mentioned FL solutions. The results show that our scheme not only resists poisoning attacks and avoids single point of failure but also achieves high privacy and computational efficiency.

III. PRELIMINARIES

This section briefly introduces the knowledge of federated learning, poisoning attacks and fully homomorphic encryption.

A. Federated Learning

In a standard federated learning setting, suppose we have a central server and n clients $\{C_1, C_2, \ldots, C_n\}$, each client has a local dataset $D_j, j = 1, 2, 3, \ldots, n$. We use $D = \{D_1, D_2, \ldots, D_n\}$ to denote the joint dataset. The n clients aim to cooperatively train a global model w without leaking their local data. As shown in Fig. 1, in the i-th iteration, the client C_j uses the local dataset and global model w_{i-1} received from the central server to train the local model. The aim of the training is to optimize the objective function described by Eq. 1.

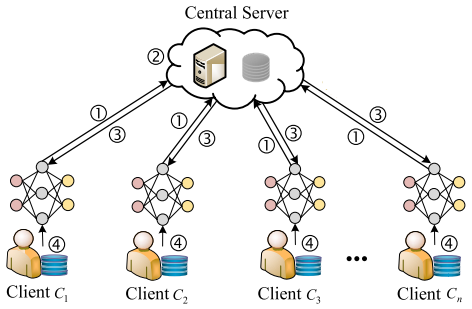


Fig. 1. Framework of FL.

$$F(x, w, y) = \min_{w} \mathbb{E}_{(x,y)\sim\widetilde{D}} L(x, w, y), \tag{1}$$

where x is the training data, y is the label, $L(x, w, y)$ is the empirical loss function and $\widetilde{D}$ is the distribution of the clients' data. Then, the client C_j sends local model w_i^j to the central server (Step ①) [6]. The central server aggregates parameters received from n clients by Eq. 2,

$$w_i = \sum\nolimits_{j=1}^{n} \zeta_i^j w_i^j, \tag{2}$$

where $\zeta_i^j = \frac{|D_j|}{|D|}$ and $\sum_{j=1}^{n} \zeta_i^j = 1$ (Step ②).

The central server sends the global model w_i to all clients (Step ③), and then each client updates local model $w_{i+1}^j = w_i^j - \alpha * \mathbf{g}_i^j$ (Step ④), where α is local learning rate.

B. Poisoning Attacks

As we all know, federated learning is vulnerable to poisoning attacks [8], [9], [15]. In the poisoning attacks, an adversary controls κ clients to manipulate local models, which ultimately affects the accuracy of the global model w. Poisoning attacks can be divided into targeted attacks [10]–[12] and untargeted attacks [7] according to the goals of the adversaries. Targeted attacks, such as Scaling attack, only target one or several data categories in the dataset, while maintaining the accuracy of other categories of data. Untargeted attacks, such as Krum attack and Trim attack, are undifferentiated attacks, which aim to reduce the accuracy of all data categories.

In order to reduce the accuracy of the global model w, data poisoning attacks [9] and model poisoning attacks [7], [11] are usually launched according to the abilities of the adversaries. In data poisoning attacks (i.e., label flipping attack), the adversaries indirectly poison the global model by poisoning the local data of the devices. In model poisoning attacks, the adversaries can directly manipulate and control the model updates of communication between the devices and the server, which

directly affects the accuracy of the global model. Therefore, model poisoning attacks usually have a worse impact on FL than data poisoning attacks.

C. Fully Homomorphic Encryption

Homomorphic Encryption (HE) [20], [21] is a cryptographic technique based on mathematical computation. HE satisfies the property that the addition or multiplication over plaintext is equivalent to the corresponding operation over ciphertext. Fully homomorphic encryption [35] is a cryptographic function that satisfies both additive and multiplicative homomorphisms and can be added and multiplied any times.

A typical FHE scheme is Cheon-Kim-Kim-Song (CKKS) [24] which allows the encryption of float numbers and vectors. CKKS has its unique encoding, decoding, and rescaling mechanisms. Specifically, for a positive integer M and a vector $\mathbf{z}$, let $\Phi_M(X)$ be the M-th cyclotomic polynomial of degree $N = \phi(M)$, the ring $\mathcal{R} = \mathbb{Z}[X]/(\Phi_M(X))$, $S = \mathbb{R}[X]/(\Phi_M(X))$ and the space $\mathbb{H} = \{(z_j)_{j\in\mathbb{Z}_M^*} : z_j = \overline{z_{-j}}, \forall j \in \mathbb{Z}_M^*\} \subseteq \mathbb{C}^N$, where $\mathbb{Z}_M^* = \{x \in \mathbb{Z}_M : gcd(x, M) = 1\}$ is the multiplicative group of units in $\mathbb{Z}_M$ and z_j is an element in the vector $\mathbf{z}$. Let $\mathcal{T}$ be a subgroup of $\mathbb{Z}_M^*$ satisfying $\mathbb{Z}_M^*/\mathcal{T} = \{\pm 1\}$. The decoding procedure first transforms a plaintext polynomial $m(X) \in \mathcal{R}$ into a complex vector $(z_j)_{j\in\mathbb{Z}_M^*} \in \mathbb{H}$ by the canonical embedding map $\sigma : S \mapsto \mathbb{C}^N$ and then sends it to a vector $(z_j)_{j\in\mathcal{T}}$ using the natural projection $\pi : \mathbb{H} \to \mathbb{C}^{N/2}$. And the encoding mechanism encodes a vector $(z_j)_{j\in\mathcal{T}}$ into a polynomial $m(X) \in \mathcal{R}$ through σ^{-1} and π^{-1}, where σ^{-1} and π^{-1} are the inverses of σ and π, respectively. Besides, in CKKS, multiplication operations change the level of ciphertext, and the rescaling mechanism is designed to avoid this. Specifically, given $p > 0$ which is used as a base for scaling in approximate computation, a modulus q_0 and a level $0 < l < L'$, a ciphertext of level l is a vector in $\mathcal{R}_{q_l}^k$ for a fixed integer k, where $q_l = p^l \cdot q_0$. Rescaling mechanism changes a ciphertext at level l to a ciphertext at level l' satisfying $0 < l' < l$. Finally, CKKS consists of the following algorithms:

- **KeyGen**$(1^\lambda) \to (sk, pk, evk)$. For a security parameter λ, this algorithm generates a secret key sk, a public key pk, and an evaluation key evk.
- **Ecd**$(\mathbf{z}, s) \to m$. For a $(N/2)$-dimensional vector $\mathbf{z} = (z_j)_{j\in\mathcal{T}}$ and a scaling factor s, this encoding algorithm encodes $\mathbf{z}$ into a polynomial $m = \sigma^{-1} \circ \pi^{-1}(\mathbf{z} \cdot s)$.
- **Enc**$(m, pk) \to c$. For a given polynomial $m \in \mathcal{R}$, this encryption algorithm encrypts m by using the public key pk and outputs a ciphertext $c \in \mathcal{R}_{q_{L'}}^k$.
- **Add**$(c_1, c_2) \to \hat{c}$. For given ciphertexts c_1 and c_2, this algorithm outputs $\hat{c} = c_1 \oplus c_2$.
- **Mult**$(c_1, c_2, evk) \to \widetilde{c}$. For a pair of ciphertexts c_1 and c_2, this algorithm outputs a ciphertext $\widetilde{c} = c_1 \otimes c_2 \in \mathcal{R}_{q_l}^k$ using evk.
- **RS**$(c) \to c'$. For a ciphertext $c \in \mathcal{R}_{q_l}^k$ at level l and a lower level $l' < l$, this rescaling procedure outputs the ciphertext $c' \leftarrow \lfloor \frac{q_{l'}}{q_l} c \rceil$ in $\mathcal{R}_{q_{l'}}^k$ where $\lfloor x \rceil$ denotes the nearest integer to a real number x.

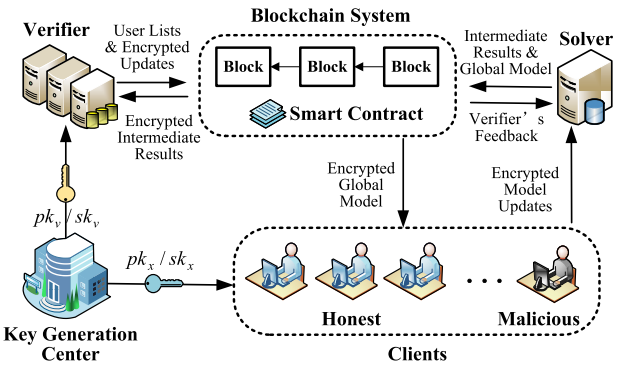


Fig. 2. System model.

- **Dec**$(c, sk) \to m$. For a ciphertext c, this algorithm outputs a polynomial m using the secret key sk.
- **Dcd**$(m, s) \to \mathbf{z}$. For an input polynomial $m \in \mathcal{R}$, this decoding procedure outputs the vector $\mathbf{z} = \pi \circ \sigma(s^{-1} \cdot m)$.

IV. Problem Formulation

In this section, we formalize the system model, problem definition, threat model, and design goals, respectively.

A. System Model

At a high level, as shown in Fig. 2, our PBFL consists of five entities: *Key Generation Center*, *Clients*, *Solver*, *Verifier* and *Blockchain System*. The role of each entity is shown as follows.

- *Key Generation Center (KGC)*: A trusted institution that generates and distributes public/private key pairs for clients and *Verifier*.
- *Clients*: As data owners, clients have public/private key pairs (pk_x, sk_x) provided by KGC and aim to benefit from the common global model.
- *Solver*: A central server with a small, clean dataset D_0 is responsible for aggregating all the gradients submitted by the clients.
- *Verifier*: A non-colluding central server and performs the calculations cooperatively with *Solver*. *Verifier* has a pair of public/private keys (pk_v, sk_v) generated by KGC.
- *Blockchain System*: To avoid selfish behaviors, the central servers need to put a deposit on SC to obtain the potential penalty. In addition, the results need to be uploaded to blockchain for a transparent computation process.

The entities in our system perform the following steps. First, each client normalizes and encrypts his/her local gradients using pk_v, and then sends them to *Solver*. *Solver* and *Verifier* communicate over multiple rounds to establish a user list without compromising privacy, in which the clients normalize gradients honestly. And the communication process needs to be recorded on blockchain. *Solver* then aggregates the gradients of the clients in the user list and gets a global model encrypted with pk_x, which are saved to blockchain. Finally, each client gets the global model from blockchain. Note that *Solver*, *Verifier* and the clients require to pay deposits to smart contract, and we omit the process in Fig. 2.

B. Problem Definition

Instead of a traditional security aggregation scenario, we consider a federated learning scenario with κ malicious clients in a total of n clients. First, we define the knowledge and capabilities of the κ malicious clients as follows:

- A malicious client may keep his own toxic data, but cannot access the local data of other honest clients.
- Malicious clients can get the encrypted global model and decrypt it to get the information. However, local model updates uploaded by a single honest client cannot be observed.
- Malicious clients can collude and share a common goal to amplify the impact of their malicious attacks.
- Malicious clients can launch either targeted attacks or untargeted attacks.

According to the assumptions about the knowledge and behaviors of malicious clients, it is obvious that the malicious clients can redirect the global model to wrong directions, rendering the efforts of honest clients useless. That is, the model aggregation we described in Eq. 2 will be defined by Eq. 3.

$$w_{i+1} = \sum_{j=1}^{n-\kappa} \zeta_i^j w_i^j - \sum_{l=1}^{\kappa} \zeta_i^l w_i^l. \tag{3}$$

Note that even a single malicious client can affect the accuracy of the global model in some cases. Due to the poisoning attacks of one or more malicious clients, the accuracy and reliability of the global model are often greatly reduced, which leads to the distrust of the global model on honest clients.

C. Threat Model

KGC is a trusted third party. *Solver* and *Verifier* are two non-colluding and "curious but honest" servers, which means that they honestly execute the established protocols but may be curious to deduce some sensitive information. In this paper, we consider two types of clients. Specifically, honest clients who aim to benefit from the global model upload true gradients trained on their local datasets. On the contrary, malicious clients upload elaborately malicious gradients and attempt to reduce the accuracy of the global model. The potential threats caused by the above entities are shown as follows:

1) Poisoning Attacks: The goal of malicious clients is to influence the performance of the global model without being detected. A malicious client can launch poisoning attacks in many ways. For example, he/she changes the labels of the data and uploads the gradients trained on the toxic data.

2) Data Leakage: Since the gradient is a mapping of the clients' local data, if the clients upload the plaintext gradients directly, the attackers can infer or obtain the original information of honest clients to a certain extent, which leads to data leakage of clients.

3) Inference Attacks: In our scheme, *Solver* and *Verifier* exchange some intermediate results for the collaboration to complete the aggregation of local updates. Therefore, they may try to infer sensitive information from the intermediate results.

TABLE II
TABLE OF MAIN NOTATIONS

Notation	Description
D_j	Training datasets of the client C_j
D_0	The trusted root dataset collected by *Solver*
w_i^j	Local model of the client C_j in the i-th iteration
α	Local learning rate
β	Global learning rate
$\mathbf{g}_i^j$	Gradient of the client C_j in the i-th iteration
$\widetilde{\mathbf{g}}_i^j$	Normalization of $\mathbf{g}_i^j$
$\mathcal{S}_i^j$	Score of the client C_j in the i-th iteration
cs_i^j	Cosine similarity between $\mathbf{g}_i^j (j \neq 0)$ and $\mathbf{g}_i^0$
b	Batch size
e	Global iterations
r	Local iterations
$\mathcal{C}$	User list in which the clients normalize honestly
pk_x/sk_x	Public/private key pair of clients
pk_v/sk_v	Public/private key pair of *Verifier*
$[\![\mathbf{g}_i^j]\!]_{pk_x}$	Encryption of the gradient $\mathbf{g}_i^j$ using pk_x

D. Design Goals

We aim to design a blockchain-based, privacy-preserving FL scheme that can resist poisoning attacks, reduce computational overheads, and provide privacy guarantees. At the same time, our scheme should achieve the same or almost the same accuracy as FedAvg or FedSGD. Specifically, we are committed to achieving the following goals:

- **Robustness.** Our scheme should be robust against malicious attacks, which means that the accuracy of the global model should not be affected by malicious clients.
- **Privacy.** We aim to protect the clients' data from being compromised. Neither a third party nor the malicious parties can get or infer the original information of clients.
- **Efficiency.** Our scheme should reduce the computational overheads caused by encryption in traditional privacy-preserving FL schemes.
- **Accuracy.** Our scheme should not sacrifice accuracy while preserving privacy and resisting malicious attacks. The accuracy of the global model should be as close as possible to that trained by FedSGD.
- **Reliability.** All the behaviors should be recorded to prevent the malicious participants from denial.

V. DESIGN OF OUR SCHEME

In this section, we first introduce the technical overview, and then describe the concrete construction of our PBFL. The main notations are shown in TABLE II.

A. Technical Overview

Traditional privacy-preserving FL schemes commonly use Paillier as a means of privacy protection. However, as we mentioned in Section I, this is not efficient. We conducted an experiment and compared the time taken by CKKS and Paillier to encrypt and decrypt vectors, as shown in Fig. 3, where the length of the vector (i.e., the number of elements contained in the vector) ranges from 100 to 500. The results show that CKKS is more efficient and more suitable for dealing with large-scale vectors and network models with many parameters

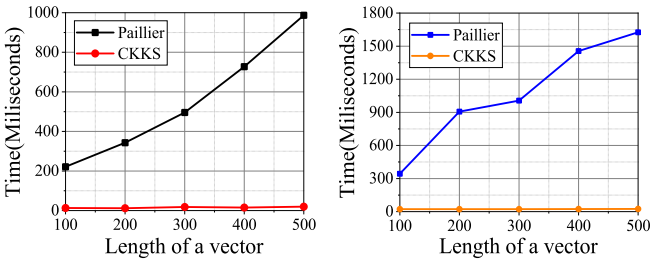


Fig. 3. The cost time of encryption and decryption between Paillier and CKKS.

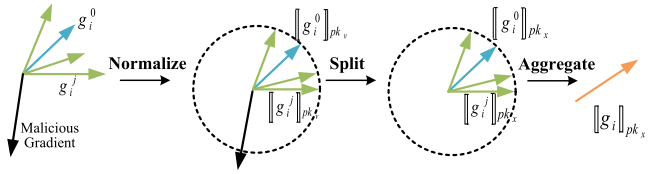


Fig. 4. Technical overview.

than Paillier. Therefore, we use CKKS to encrypt the local gradients in parallel to improve computational efficiency.

Besides, we also need to consider that a malicious client may disrupt the training process by sending malicious gradients, and the "honest but curious" central server may infer clients' sensitive information. Therefore, in this paper, we aim to protect the data privacy of clients and counteract the adverse efforts of poisoning attacks, making the whole training process as if the malicious clients do not exist. To do this, we represent our aggregation operation as Eq. 4, where the set $\mathcal{C}$ is defined as the collection of honestly normalized clients selected according to our rule and $|\mathcal{C}|$ is the number of clients in set $\mathcal{C}$.

$$w_{i+1} = \sum_{j=1}^{|\mathcal{C}|} \zeta_i^j w_i^j. \tag{4}$$

Fig. 4 shows the overview of our scheme. First, we rely on fully homomorphic encryption technology CKKS [24] to encrypt local gradients for privacy protection. Each client encrypts and normalizes his/her local updates and sends them to *Solver*. Then we put the clients who upload the normalized gradients into the set $\mathcal{C}$, and use a cosine similarity-based strategy to identify honest and malicious gradient vectors. Specifically, consistent with the settings in FLTrust [16], we store a small, clean root dataset D_0 in *Solver* and maintain a model w^0 based on it. If the cosine similarity between the gradient $\mathbf{g}_i^0$ and $\mathbf{g}_i^j$ uploaded by the client C_j is less than zero, which means that the client C_j is a malicious one and will make a negative contribution to the aggregation. Therefore, *Solver* chooses to discard the gradient $\mathbf{g}_i^j$ of the client C_j at the i-th iteration.

As we know, gradients can be regarded as vectors with magnitudes and directions. Undoubtedly, the magnitudes of the gradients also affect the global model. For example, a malicious client tends to upload local gradients with larger magnitudes in order to amplify his/her impacts. So we normalize the gradients before sending them to *Solver*. However, malicious clients may upload gradients without normalization. To solve this problem, we introduce *Verifier*, a non-colluding central sever with *Solver*. Through the communication between *Solver* and *Verifier*, the set of honestly normalized clients and local updates encrypted with pk_x are obtained by *Solver*. Finally, *Solver* aggregates local updates to get the global model.

To prevent the malicious behaviors of central servers and reduce the risk of single point of failure, it should be noted that the user list $\mathcal{C}$ and the intermediate results calculated by *Solver* and *Verifier* cooperatively need to be saved to blockchain. Specifically, *Solver* and *Verifier* pay deposits to smart contract, which encourages them to make correct calculations. In addition, the intermediate results and the encrypted global model of each iteration of training are saved to blockchain for timely backtracking in case of the failure of central servers, which increases the reliability of our scheme.

B. Construction of PBFL

PBFL consists of three processes, local computation, normalizing judgment, and model aggregation. In the following, we describe the details of the processes.

1) Local Computation: Local computation consists of four parts: local training, normalization, encryption, and model update. We describe the processes of **`Local_Computation`** in Algorithm 1. In the following, we give the details about these parts.

a) Local training: In the i-th iteration of the training, each client C_j participating in FL uses local dataset D_j, $j \in [1, n]$ and local model w_i^j to get the local gradient which is described as Eq. 5.

$$\mathbf{g}_i^j = \nabla L(w_i^j, D_j), \tag{5}$$

where $L(w_i^j, D_j)$ is the experical loss function, ∇ is the derivation operation.

Algorithm 1 **`Local_Computation`**

Input: n clients $\{C_1, C_2, \ldots C_n\}$ with local training datasets $D = \{D_1, D_2, \ldots, D_n\}$, batch size b, local learning rate α, number of local iterations r.
Output: Encrypted local gradients.

for *each* $j \in [C_1, C_2, \ldots C_n]$ **do**
  Get global model $[\![w_{i-1}]\!]_{pk_x}$ from blockchain;
  Decrypt $[\![w_{i-1}]\!]_{pk_x}$ using sk_x to get w_{i-1};
  $\mathbf{g}_i^j$ =**`ModelUpdate`**$(w_{i-1}, D_j, b, \alpha, r)$;
  $\widetilde{\mathbf{g}}_i^j = \mathbf{g}_i^j / ||\mathbf{g}_i^j||$;
  Get encrypted gradient $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$ using pk_v;
  Send $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$ to *Solver*;
return $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$.

b) Normalization: Our aggregation rule is based on the cosine similarity strategy. In order to make our aggregation rule work in ciphertext, we normalize the local gradients before encryption. Specifically, we treat the gradients as directional vectors. For the client C_j, we use Eq. 6 to normalize

the local gradients,

$$\widetilde{\mathbf{g}}_i^j = \mathbf{g}_i^j / \| \mathbf{g}_i^j \|, \tag{6}$$

where $\widetilde{\mathbf{g}}_i^j$ is the unit vector. Each client needs to normalize the local gradients before encryption. First, the normalization operation allows our aggregation rule to be applied directly to ciphertext without any changes. Because we convert the cosine similarity into the inner product of vectors due to normalization. Second, the magnitudes of the vectors are same, which alleviates the effects of the malicious gradients. This is based on the intuition that malicious clients tend to upload local gradients with a larger magnitudes in order to amplify their impacts.

c) Encryption: The client C_j encrypts the local gradient $\widetilde{\mathbf{g}}_i^j$ using the public key pk_υ before uploading it to *Solver*. We recommend CKKS [24] scheme to encrypt the local gradients. The gradients are usually signed floating-point numbers. If we use another encryption scheme such as Paillier [20], we first need to quantize and clip the gradient values, and then encrypt each value individually, which undoubtedly incurs high computation overheads. Thus, we use CKKS to encrypt the local gradients.

In our scheme, we treat the gradients $\widetilde{\mathbf{g}}_i^j$ of each layer as vectors and encrypt the gradients layer by layer. Specifically, each client uses *Verifier*'s public key pk_υ to encrypt the gradients of the different layers directly. If the length of the vector $\widetilde{\mathbf{g}}_i^j$ is too long, we encrypt it multiple times. For the rest of the vectors, we just need to encrypt them once.

d) Model update: The client C_j downloads the latest global model from blockchain. Then, the client C_j decrypts it with his/her private key sk_x to get the plaintext global model w_{i-1}. After that, the local model w_i^j is updated with Eq. 7, where α is the learning rate. When the objective function of clients converges, the whole training process is over. Otherwise, the clients start the next iteration.

$$w_i^j \leftarrow w_{i-1} - \alpha \mathbf{g}_i^j. \tag{7}$$

By calling **ModelUpdate** in Algorithm 2, we describe the details for updating the local models.

Algorithm 2 **ModelUpdate**

Input: Local model w, local dataset D, batch size b, local learning rate α, local iterations r.
Output: Gradient $\mathbf{g}_r$.

$w_0 \leftarrow w$;
for *each* $l \in [1, r]$ **do**
  Randomly sample a batch D_b from D;
  $\mathbf{g}_l \leftarrow \nabla L(D_b, w_{l-1})$;
  $w_l \leftarrow w_{l-1} - \alpha \mathbf{g}_l$;
return $\mathbf{g}_r$.

2) Normalizing Judgement: Solver needs to determine if the gradients are truly normalized after receiving the gradients from the clients. Considering the privacy requirements of our scheme, we implement the security judgment of the gradients by calling **NormJudge** in Algorithm 3 collaboratively.

As shown in Algorithm 3, upon receiving encrypted local updates $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon}$, *Solver* computes $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon} \odot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon}$ and sends it to *Verifier*, where "$\odot$" is the inner product. Suppose a n^*-dimensional vector $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon}$ is encrypted into a ciphertext, we next describe the details about computing the inner product.

Algorithm 3 **NormJudge**

Input: *Solver* holds $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon}$ and *Verifier* holds sk_υ.
Output: User list $\mathcal{C}$.

Solver:
Send $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon} \odot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon}$ to blockchain;
Verifier:
Initialize set $\mathcal{C} \leftarrow \varnothing$;
for *each* $j \in [1, n]$ **do**
  Download $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon} \odot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon}$ from blockchain;
  Decrypt $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon} \odot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon}$ using sk_υ to get len_i^j;
  if $len_i^j == 1$ **then**
    Add the client C_j to $\mathcal{C}$;
Send $\mathcal{C}$ to blockchain;
return $\mathcal{C}$.

Specifically, CKKS works with polynomials because it provides a good trade-off between security and efficiency when compared with standard computations on vectors. Once the message is encrypted into a couple of polynomials, CKKS provides several operations that can be performed on it, such as addition, multiplication, and rotation. Suppose a n^*-dimensional vector $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon}$ is represented by $[\![p_1, p_2, \ldots, p_{n^*}]\!]_{pk_\upsilon}$, to get the inner product between vectors, we first multiply two encrypted vectors to get $[\![p_1^2, p_2^2, \ldots, p_{n^*}^2]\!]_{pk_\upsilon}$. We then rotate the vector $[\![p_1^2, p_2^2, \ldots, p_{n^*}^2]\!]_{pk_\upsilon}$ to get $[\![p_2^2, p_3^2, \ldots, p_{n^*}^2, p_1^2]\!]_{pk_\upsilon}$, and add these two vectors [29], [30]. Repeating the rotation and addition operations $(n^* - 1)$ times, we get $[\![r_1, r_2, r_3, \ldots, r_{n^*}]\!]_{pk_\upsilon}$ eventually, where $r_1 = p_1^2 + p_2^2 + \cdots + p_{n^*}^2$. Finally, we multiply two vectors $[\![r_1, r_2, r_3, \ldots, r_{n^*}]\!]_{pk_\upsilon}$ and $[1, 0, 0, \ldots, 0]$ to get $[\![r_1]\!]_{pk_\upsilon}$, where $[\![r_1]\!]_{pk_\upsilon} = [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon} \odot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon}$.

3) Model Aggregation: After receiving the user list $\mathcal{C}$ from *Verifier*, *Solver* securely aggregates the gradients uploaded by clients who are in set $\mathcal{C}$. Specifically, *Solver* and *Verifier* first securely execute a protocol that enables *Solver* to obtain the intermediate results and the local model updates encrypted with pk_x. Then *Solver* gets the encrypted global model according to the predefined aggregation rule and uploads it to blockchain.

a) 2-Party computation: To aggregate local updates, *Solver* first computes the cosine similarity between $[\![\widetilde{\mathbf{g}}_i^0]\!]_{pk_\upsilon}$ and $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_\upsilon}$ and communicates with *Verifier*. Specifically, our aggregation rule relies on a trusted root dataset D_0 and its corresponding model w^0, which are used to determine the more "promising" direction of the global model update. Local model updates that are more similar to the direction of $\mathbf{g}_i^0$ have higher weights while being aggregated. As in previous work [16], *Solver* can collect a trusted root dataset D_0 by manually marking. For example, Google can enlist its employees

to type with Gboard to create root dataset in the federated task of the next-word prediction. In section VII, we show that we only need a small root dataset D_0 (such as 200 data points), and the distribution of the dataset can be different from that of the clients. Therefore, the costs of collecting the trusted root dataset D_0 and manual labeling are usually affordable for *Solver*.

$$\begin{aligned}[\![cs_i^j]\!]_{pk_v} &= [\![\widetilde{\mathbf{g}}_i^0]\!]_{pk_v} \odot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v} \\ &= [\![p_1q_1 + p_2q_2 + \cdots + p_{n^*}q_{n^*}]\!]_{pk_v}. \\ s.t., ||\widetilde{\mathbf{g}}_i^0|| &= ||\widetilde{\mathbf{g}}_i^j|| = 1. \end{aligned} \tag{8}$$

Therefore, in the i-th iteration, *Solver* trains on dataset D_0 to get the gradient update $\mathbf{g}_i^0$, and then uses $\widetilde{\mathbf{g}}_i^0 = \mathbf{g}_i^0/||\mathbf{g}_i^0||$ to normalize $\mathbf{g}_i^0$. Then *Solver* encrypts it to get the gradient $[\![\widetilde{\mathbf{g}}_i^0]\!]_{pk_v}$. Then we calculate the cosine similarity between these two vectors in Eq. 8 where the client C_j gets $\widetilde{\mathbf{g}}_i^j = [p_1, p_2, \ldots, p_{n^*}]$ and *Solver* gets $\widetilde{\mathbf{g}}_i^0 = [q_1, q_2, \ldots, q_{n^*}]$. And the encrypted gradients are obtained as $[\![p_1, p_2, \ldots, p_{n^*}]\!]_{pk_v}$ and $[\![q_1, q_2, \ldots, q_{n^*}]\!]_{pk_v}$. Due to normalization, the cosine similarity between the two vectors can be transformed to inner product.

$$\mathcal{S}_i^j = ReLU(cs_i^j). \tag{9}$$

cs_i^j measures the similarity between $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$ and $[\![\widetilde{\mathbf{g}}_i^0]\!]_{pk_v}$. The negative value of cs_i^j means that the direction of the local gradient $\widetilde{\mathbf{g}}_i^j$ is opposite to the direction of $\widetilde{\mathbf{g}}_i^0$, which negatively affects the global model. A typical solution is to discard the malicious gradients during aggregation, thereby minimizing the effects of these gradients as much as possible. From this point of view, the ReLU function is used to constrain the gradients in set $\mathcal{C}$. Therefore, in the i-th iteration, the score $\mathcal{S}_i^j$ of the client C_j is defined by Eq. 9 and the ReLU function is defined by Eq. 10.

$$ReLU(x) = \begin{cases} x. & if\ x > 0 \\ 0. & if\ x \le 0 \end{cases} \tag{10}$$

From Eq. 10, ReLU function actually returns the larger one between x and 0. Note that cs_i^j is computed from two encrypted vectors. Therefore, inspired by previous work [34], we next give the operation of the ReLU function under ciphertext.

Algorithm 4 `Max`

Input: $[\![a]\!], [\![b]\!] \in [0, 1], d \in \mathbb{N}$.
Output: an approximate value of max($[\![a]\!]$,$[\![b]\!]$).

$x = ([\![a]\!] + [\![b]\!])/2$ and $y = ([\![a]\!] - [\![b]\!])/2$;
$a_0 \leftarrow y^2$;
$b_0 \leftarrow y^2 - 1$;
for *each* $n \in [0, d-1]$ **do**
  $a_{n+1} \leftarrow a_n(1 - \frac{b_n}{2})$;
  $b_{n+1} \leftarrow b_n^2(\frac{b_n - 3}{4})$;
$z = a_d$;
return $x + z$.

The scheme [34] shows a numerical method for the comparison of homomorphic ciphers in the range of [0, 1]. However, the cosine similarity between the two vectors falls on $[-1, 1]$, which prevents us from directly applying the comparison method to the ciphertext comparison of $[\![cs_i^j]\!]$ and $[\![0]\!]$. One of the solutions is to convert the cosine similarity to the allowable range. To do this, we first add $[\![1]\!]$ to $[\![cs_i^j]\!]$, so that the plaintext of the result falls on $[0, 2]$, and then we multiply the result by $\frac{1}{2}$. Thus, the final result meets the requirement. Note that the value 0 is corresponding to the value $\frac{1}{2}$ through our transformation. Therefore, we transform the ReLU function to ReLU′ which compares the final result with $[\![\frac{1}{2}]\!]$ and *ReLU′* is implemented using Eq. 11. We describe the homomorphic comparison method by calling `Max` in Algorithm 4.

$$ReLU'([\![cs_i^j]\!]) = \begin{cases} \frac{1}{2}([\![cs_i^j]\!] + [\![1]\!]). & if\ \frac{1}{2}([\![cs_i^j]\!] + [\![1]\!]) > [\![\frac{1}{2}]\!] \\ [\![1/2]\!]. & if\ \frac{1}{2}([\![cs_i^j]\!] + [\![1]\!]) \le [\![\frac{1}{2}]\!] \end{cases} \tag{11}$$

After getting all the scores of clients, *Solver* works with *Verifier* to get the values needed for the model aggregation without revealing privacy. Specifically, to get the original value of the score, *Solver* sets $[\![\mathcal{S}_i^j]\!]'_{pk_v} = 2 \cdot [\![\mathcal{S}_i^j]\!]_{pk_v} - [\![1]\!]_{pk_v}$. Then *Solver* computes $[\![sum]\!]_{pk_v} = \sum_{f=1}^{|\mathcal{C}|} [\![\mathcal{S}_i^f]\!]'_{pk_v}$ and *Solver* randomly selects a n^*-dimensional vector V^{n^*} and $\widetilde{V}$ to get $V^{n^*} \cdot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$ and $\widetilde{V} \cdot [\![\mathcal{S}_i^j]\!]'_{pk_v}$. Finally, *Solver* sends $[\![sum]\!]_{pk_v}$, $V^{n^*} \cdot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$ and $\widetilde{V} \cdot [\![\mathcal{S}_i^j]\!]'_{pk_v}$ to blockchain. *Verifier* gets them and decrypts with sk_v, then *Verifier* encrypts $V^{n^*} \cdot \widetilde{\mathbf{g}}_i^j$ and $\widetilde{V} \cdot \mathcal{S}_i^{j'}$ using pk_x, and sends sum, $[\![V^{n^*} \cdot \widetilde{\mathbf{g}}_i^j]\!]_{pk_x}$ and $[\![\widetilde{V} \cdot \mathcal{S}_i^{j'}]\!]_{pk_x}$ to blockchain. By calling `GetParam` in Algorithm 5, we describe the above process.

Algorithm 5 `GetParam`

Input: *Solver* holds $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$, $[\![\widetilde{\mathbf{g}}_i^0]\!]_{pk_v}$, user list $\mathcal{C}$, a sufficiently large number d; *Verifier* holds the private key sk_v.
Output: The values needed for aggregation.

Solver:
for *each* $j \in \mathcal{C}$ **do**
  $[\![cs_i^j]\!]_{pk_v} = [\![\widetilde{\mathbf{g}}_i^0]\!]_{pk_v} \odot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$;
  $[\![\mathcal{S}_i^j]\!]_{pk_v} = \mathbf{Max}(([\![cs_i^j]\!]_{pk_v} + [\![1]\!]_{pk_v}) \cdot \frac{1}{2}, [\![\frac{1}{2}]\!]_{pk_v}, d)$;
  $[\![\mathcal{S}_i^j]\!]'_{pk_v} = 2 \cdot [\![\mathcal{S}_i^j]\!]_{pk_v} - [\![1]\!]_{pk_v}$;
$[\![sum]\!]_{pk_v} = \sum_{f=1}^{|\mathcal{C}|} [\![\mathcal{S}_i^f]\!]'_{pk_v}$;
Randomly select a n^*-dimensional vector V^{n^*} and $\widetilde{V}$;
Send $[\![sum]\!]_{pk_v}$, $V^{n^*} \cdot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$ and $\widetilde{V} \cdot [\![\mathcal{S}_i^j]\!]'_{pk_v}$ to blockchain;
Verifier:
Download $[\![sum]\!]_{pk_v}$, $V^{n^*} \cdot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$ and $\widetilde{V} \cdot [\![\mathcal{S}_i^j]\!]'_{pk_v}$ from blockchain and decrypt them with sk_v to get sum, $V^{n^*} \cdot \widetilde{\mathbf{g}}_i^j$ and $\widetilde{V} \cdot \mathcal{S}_i^{j'}$;
Encrypt $V^{n^*} \cdot \widetilde{\mathbf{g}}_i^j$ and $\widetilde{V} \cdot \mathcal{S}_i^{j'}$ using pk_x to get $[\![V^{n^*} \cdot \widetilde{\mathbf{g}}_i^j]\!]_{pk_x}$ and $[\![\widetilde{V} \cdot \mathcal{S}_i^{j'}]\!]_{pk_x}$;
Send sum, $[\![V^{n^*} \cdot \widetilde{\mathbf{g}}_i^j]\!]_{pk_x}$ and $[\![\widetilde{V} \cdot \mathcal{S}_i^{j'}]\!]_{pk_x}$ to blockchain;
return sum, $[\![V^{n^*} \cdot \widetilde{\mathbf{g}}_i^j]\!]_{pk_x}$ *and* $[\![\widetilde{V} \cdot \mathcal{S}_i^{j'}]\!]_{pk_x}$.

In previous work [38], the double-server architecture is used to compare ciphertext, which can also be implemented in our scheme. However, if we implement the ReLU function under the ciphertext in **GetParam** by using double-server architecture and masking cs_i^j, the following results may occur: 1) The double-server architecture can infer the sign of cs_i^j, which may be detrimental to clients. 2) The communication rounds of two servers may increase. In addition, the previous work [34], [40], [42] give outstanding contributions, which make it more convenient to perform ciphertext operations. Therefore, in our scheme, *Solver* uses **Max** algorithm to implement the ReLU function under the ciphertext locally, which does not disclose private information of clients, and reduces the communication overheads.

b) Aggregation: After getting sum, $[\![V^{n^*} \cdot \widetilde{\mathbf{g}}_i^j]\!]_{pk_x}$ and $[\![\widetilde{V} \cdot \mathcal{S}_i^{j'}]\!]_{pk_x}$ from blockchain, *Solver* first multiplies $[\![V^{n^*} \cdot \widetilde{\mathbf{g}}_i^j]\!]_{pk_x}$ and $[\![\widetilde{V} \cdot \mathcal{S}_i^{j'}]\!]_{pk_x}$ with $1/V^{n^*}$ and $1/\widetilde{V}$ to get $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_x}$ and $[\![\mathcal{S}_i^j]\!]'_{pk_x}$. Then, *Solver* aggregates the gradients weighted by their own scores without revealing privacy. We use Eq. 12 to aggregate local updates in the i-th epoch.

$$\begin{aligned}[\![\widetilde{\mathbf{g}}_i]\!]_{pk_x} &= \frac{1}{sum}\sum_{j=1}^{|\mathcal{C}|}[\![\mathcal{S}_i^j]\!]'_{pk_x} \cdot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_x} \\ &= \frac{1}{sum}\sum_{j=1}^{|\mathcal{C}|} ReLU([\![\widetilde{\mathbf{g}}_i^0]\!]_{pk_x} \odot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_x}) \cdot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_x}. \end{aligned} \tag{12}$$

Algorithm 6 **PBFL**

Input: n clients $\{C_1, C_2, \ldots C_n\}$ with local training datasets $D = \{D_1, D_2, \ldots, D_n\}$, batch size b, local learning rate α, global learning rate β, global iterations e, local iterations r, a sufficiently large d.

Output: Encrypted global model.

KGC distributes key pairs to clients and *Verifier*;
Solver initializes model w_0 and sends it to blockchain;
for *each* $i \in [1, e]$ **do**
  $[\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}$ = **Local_Computation**(D, b, α, r);
  Solver and Verifier:
  $\mathcal{C}$ = **NormJudge**$([\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}, sk_v)$;
  sum, $[\![V^{n^*} \cdot \widetilde{\mathbf{g}}_i^j]\!]_{pk_x}$, $[\![\widetilde{V} \cdot \mathcal{S}_i^{j'}]\!]_{pk_x}$ = **GetParam** $([\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_v}, [\![\widetilde{\mathbf{g}}_i^0]\!]_{pk_v}, \mathcal{C}, d, sk_v)$;
  Solver:
  for *each* $c \in \mathcal{C}$ **do**
    $[\![\widetilde{\mathbf{g}}_i^c]\!]_{pk_x} = [\![V^{n^*} \cdot \widetilde{\mathbf{g}}_i^c]\!]_{pk_x} \cdot \frac{1}{V^{n^*}}$;
    $[\![\mathcal{S}_i^c]\!]'_{pk_x} = [\![\widetilde{V} \cdot \mathcal{S}_i^{c'}]\!]_{pk_x} \cdot \frac{1}{V}$;
  $[\![\widetilde{\mathbf{g}}_i]\!]_{pk_x} = \frac{1}{sum}\sum_{j=1}^{|\mathcal{C}|}[\![\mathcal{S}_i^j]\!]'_{pk_x} \cdot [\![\widetilde{\mathbf{g}}_i^j]\!]_{pk_x}$;
  $[\![w_i]\!]_{pk_x} = [\![w_{i-1}]\!]_{pk_x} - \beta[\![\widetilde{\mathbf{g}}_i]\!]_{pk_x}$;
  Send $[\![w_i]\!]_{pk_x}$ to blockchain;
return $[\![w_e]\!]_{pk_x}$.

In our scheme, *Solver* and *Verifier* cannot get any information except sum. And we think that the disclosure of sum is meaningless, because *Solver* and *Verifier* cannot get valuable information from sum. Besides, both *Solver* and *Verifier* are required to submit deposits to smart contract before the start of the federated task, and will be rewarded with the clients' deposits at the end of the task. Because we think that the financial incentive mechanism prompts the servers to do the correct calculation instead of submitting meaningless or malicious results. Finally, our **PBFL** is described in Algorithm 6.

VI. Security Analysis

For a specific FL task, the optimal global model w^* can be obtained by solving the following optimization problem,

$$\min_w \mathbb{E}_{(x,y)\sim\widetilde{D}} L(x, w, y),$$

where $L(x, w, y)$ is the experical loss function and $\widetilde{D}$ is the distribution of the training data. We use $F(w)$ to denote $\mathbb{E}_{(x,y)\sim\widetilde{D}} L(x, w, y)$. With the following assumptions, we prove that the error between the model learned by our scheme and the optimal global model w^* is bounded.

Assumption 1: The function $F(w)$ is μ-strongly convex and differentiable over the parameter space with $\mathcal{L} - Lipschitz$ continuous gradients. The loss function $L(x, w, y)$ is $\mathcal{L}_1 - Lipschitz$ probabilistically. For any w, $\overline{w}$, we have, [16]

$$F(\overline{w}) \geq F(w) + \nabla F(w) \odot (\overline{w} - w) + \frac{\mu}{2} \parallel \overline{w} - w \parallel^2,$$
$$\parallel \nabla F(w) - \nabla F(\overline{w}) \parallel \leq \mathcal{L} \parallel w - \overline{w} \parallel,$$

where $\nabla F(w)$ is gradient, $\boldsymbol{x} \odot \boldsymbol{y}$ is the inner product of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. For any $\delta \in (0, 1)$ and $\overline{l} = 1 - \frac{\delta}{3}$, there exists a constraint $\mathcal{L}_1$ that,

$$Pr\left\{\sup_{w \neq \overline{w}} \frac{\parallel \nabla L(x, w, y) - \nabla L(x, \overline{w}, y) \parallel}{\parallel w - \overline{w} \parallel} \leq \mathcal{L}_1\right\} \geq \overline{l}.$$

Assumption 2: The gradient of the experical loss function $L(x, w^*, y)$ at the optimal global model w^* is bounded. For any w, $h(x, w, y) = \nabla L(x, w, y) - \nabla L(x, w^*, y)$ is bounded. Morever, for any unit vector $\boldsymbol{v}$, there exist positive constants σ_1 and γ_1 such that $\nabla L(x, w^*, y) \odot \boldsymbol{v}$ is sub-exponential. And for any $w \neq w^*$ and vector $\boldsymbol{v}$, there exist positive constants σ_2 and γ_2 such that $((h(x, w, y) - \mathbb{E}[h(x, w, y)]) \odot \boldsymbol{v})/||w - w^*||$ is sub-exponential with σ_2 and γ_2. Then, let O be $e^{\frac{\sigma_2^2\xi^2}{2}}$, for any $|\xi| \leq 1/\gamma_1$, any $|\xi| \leq 1/\gamma_2$ and $\boldsymbol{B} = \{\boldsymbol{v} : \parallel \boldsymbol{v} \parallel = 1\}$, we have the following equation [16].

$$\sup_{\boldsymbol{v}\in\boldsymbol{B}} \mathbb{E}[\exp(\xi(\nabla L(x, w^*, y) \odot \boldsymbol{v}))] \leq e^{\frac{\sigma_1^2\xi^2}{2}},$$
$$\sup_{\boldsymbol{v}\in\boldsymbol{B}} \mathbb{E}\left[\exp\left(\frac{\xi((h(x, w, y) - \mathbb{E}[h(x, w, y)]) \odot \boldsymbol{v})}{\parallel w - w^* \parallel}\right)\right] \leq O.$$

Assumption 3: The root dataset D_0 and the clients' dataset $D_i (i = 1, 2 \ldots, n)$ are sampled independently from distribution $\widetilde{D}$ [16].

Next we give Theorem 1 and Theorem 2, in which Theorem 2 is built on Assumption 1- 3.

Theorem 1: Solver, Verifier, and malicious clients can get nothing about the sensitive information of honest clients.

Proof: In our scheme, blockchain exposes sum to all participants, which is the sum of the honest clients' weights.

The weights are represented by $ReLU(cs_i^j)$, where cs_i^j is the cosine similarity between the gradients of the client C_j and $\mathbf{g}_i^0$. For *Solver* and *Verifier*, they get sum, where $sum = \mathcal{S}_i^{1\prime} + \mathcal{S}_i^{2\prime} + \cdots + \mathcal{S}_i^{|\mathcal{C}|\prime}$. Since *Solver* and *Verifier* are non-colluding servers, they cannot get the exact value of $\mathcal{S}_i^j$, where $j \in [1, |\mathcal{C}|]$. For κ ($\kappa < n$) malicious clients, we assume that they conspire to achieve their goals. Thus they get $sum' = sum - (\mathcal{S}_i^{1\prime} + \mathcal{S}_i^{2\prime} + \cdots + \mathcal{S}_i^{\kappa\prime})$, where $\mathcal{S}_i^{1\prime} + \mathcal{S}_i^{2\prime} + \cdots + \mathcal{S}_i^{\kappa\prime}$ is known to the malicious clients and sum' is the sum of the honest clients' scores. When $\kappa = n-1$, the malicious clients get $\mathcal{S}_i^n$ of the honest client C_n. However, 1) the presence of $n-1$ malicious clients in n clients is not realistic, because such a federated learning task is meaningless and beyond the scope of our research. 2) Since $\mathcal{S}_i^n = ReLU(\mathbf{g}_i^n \odot \mathbf{g}_i^0)$, and $\mathbf{g}_i^0$ is computed and stored in *Solver*, the malicious clients cannot get $\mathbf{g}_i^0$. Thus they cannot obtain any valuable information about the gradients of honest client C_n. Therefore, as we described in Theorem 1, neither a third party nor malicious clients can infer sensitive information about honest clients. □

Theorem 2: The error between the model learned in the i-th epoch and the optimal global model w^* is $||w^i - w^*|| \leq (1-\rho)^i\,||w^0 - w^*|| + 12\beta\Delta_1/\rho + noise$ with probability at least $1-\delta$, where w_i is the global model in the i-th iteration and $noise$ is the error caused by CKKS.

Proof: Suppose the above assumptions hold, we use stochastic gradient descent for 1 iteration and set $\alpha = 1$ to perform the function **`ModelUpdate`** in Algorithm 2. We use β to represent global learning rate and $\beta = \mu/(2\mathcal{L}^2)$. Let r' be a positive number, $\widetilde{X}$ be $\log\left(6\sigma_2^2 r'\sqrt{|D_0|}/\gamma_2\sigma_1\delta\right)$, d' be the dimension of w, $\mathcal{L}_2 = \max\{\mathcal{L}, \mathcal{L}_1\}$, then we define ρ, Δ_1, Δ_2 as follows,

$$\rho = 1 - \left(\sqrt{1-\mu^2/(4\mathcal{L}^2)} + 24\beta\Delta_2 + 2\beta\mathcal{L}\right),$$

$$\Delta_1 = \sigma_1\sqrt{\frac{2}{D_0}}\sqrt{d'\log 6 + \log(3/\delta)},$$

$$\Delta_2 = \sigma_2\sqrt{\frac{2}{|D_0|}}\sqrt{d'\log\frac{18\mathcal{L}_2}{\sigma_2} + \frac{1}{2}d'\log\frac{|D_0|}{d'} + \widetilde{X}}.$$

Our scheme is an improvement and extension of [16] in ciphertext and blockchain, so the security analysis of our scheme can be ditectly and well applied. In [16], the error between the model learned in the i-th epoch and the optimal global model w^* is $||w^i - w^*|| \leq (1-\rho)^i\,||w^0 - w^*|| + 12\beta\Delta_1/\rho$ with probability at least $1-\delta$, where w_i is the global model in the i-th iteration. Therefore, in our scheme, we have the following formula with probability at least $1-\delta$,

$$\| w_i - w^* \| \leq (1-\rho)^i \| w^0 - w^* \| + 12\beta\Delta_1/\rho + noise,$$

where $noise$ is the error caused by CKKS.

Next we give the details about $noise$. As described in [24], For a positive integer M, let $\Phi_M(X)$ be the M-th cyclotomic polynomial of degree $N = \phi(M)$. We define a modulus q_0 and a integer p be used as a base for scaling in approximate computation, let $q_l = p^l \cdot q_0$ for any $0 < l < L'$, where L' is multiplicative depth. For a real value $\sigma = \sigma(\lambda, q_l)$, where λ is the security parameter and a positive interger h, the encoding and encryption noises are bounded by the following formula,

$$B = 8\sqrt{2}\sigma N + 6\sigma\sqrt{N} + 16\sigma\sqrt{hN}.$$

For $j = 1, 2$, let v be the upper bound of plaintext message, B_j be the upper bound of noise for a message m_j. The noise from addition operation is $A = B_1 + B_2$ and the multiplication operation is $M' = v_1B_2 + v_2B_1 + B_1B_2 + B_{mult(l)}$, where $B_{mult(l)} = P^{-1}\cdot q_l \cdot B_{ks} + B_{scale}$, $B_{ks} = 8\sigma N/\sqrt{3}$, $P = P(\lambda, q_{L'})$ is an integer and $B_{scale} = \sqrt{N/3}\cdot(3 + 8\sqrt{h})$. The noise generated by the ciphertext operation can therefore be expressed as follows.

$$noise = B + B_1 + B_2 + v_1B_2 + v_2B_1 + B_1B_2 + B_{mult(l)}.$$

More detailed proof can be found in [24]. Then, for r' and any model w such that $||w - w^*|| \leq r'\sqrt{d'}$, when $|1-\rho| < 1$, we have $\lim_{i\to\infty} ||w^i - w^*|| \leq 12\beta\Delta_1/\rho + noise$. Therefore, the error between the model learned by our scheme and the optimal global model w^* is bounded. □

TABLE III
DESCRIPTIONS OF DATASETS

Dataset	Training set	Class	Test set	Data type
MNIST	60000	10	10000	image
FashionMNIST	60000	10	10000	image

VII. PERFORMANCE ANALYSIS

We evaluate our scheme against both targeted and untargeted attacks and show the results of our experiments.

A. Experimental Setup

1) Datasets and Settings: We adopt two widely used datasets, MNIST and FashionMNIST to evaluate our scheme. The details about the datasets are shown in TABLE III. The MNIST dataset is composed of handwritten numbers from 250 different people, 50% of whom are high school students, 50% of whom are staff members of the Census Bureau, and the same proportion of handwritten digital data from the test set. The dataset contains 60, 000 training images and 10, 000 test images. If there are no special instructions, we choose 200 data as the dataset D_0, and the data distribution is balanced in the dataset D_0. The FashionMNIST dataset is similar to MNIST, with images in 10 categories, including dress, pullover, coat, and so on. The FashionMNIST has the same size as MNIST. The principle for selecting the dataset D_0 in FashionMNIST is the same as the MNIST dataset.

We implement and evaluate our scheme on a private Ethereum blockchain setup. The SC layer is developed using Solidity programming language and deployed on the private blockchain using Truffle. The private blockchain is deployed on an Intel CPU with a clock rate of 2.40 GHz with 4 cores and 2 threads per core. Besides, we use TenSEAL library which is the python version of Microsoft's Simple Encrypted Arithmetic Library (SEAL) to deploy CKKS. TenSEAL is an open-source homomorphic encryption library, and the code is available on Github (https://github.com/OpenMined/TenSEAL).

TABLE IV
MODEL SUMMARY

Layer(type)	*Ouput Shape*	*Parameters*
flatten(Flatten)	784	0
dense(Dense)	128	100480
dense_1(Dense)	10	1290

2) Poisoning Attacks: In this experiment, we consider both targeted attack and untargeted attack. For untargeted attack, we assume that the malicious clients upload arbitrary updates to manipulate the global model. And for targeted attack, we evaluate label-flipping attack. Specifically, we flip the label form l to $\mathcal{N}-l-1$ for each malicious client, where $l \in \{0, 1, \ldots, \mathcal{N}-1\}$ and $\mathcal{N}$ is the total number of the labels.

3) Evaluation Metrics: We use test accuracy and test error rate as indicators of the models trained on different datasets. Because the goal of our scheme is to improve the accuracy of the global model. FedSGD [6] is a popular method in non-adversarial settings in FL. Therefore, we evaluate FedSGD scheme without attacks as a baseline, Krum [14] scheme as a comparison, and demonstrate the results of our scheme in the presence of different numbers of malicious clients.

4) FL Settings: In our evaluation, we adopt the cross-silo setting. That is, we set the number of clients as $n = 10$ and select all clients in each iteration during the training. For model selection, we use a 3 layer Neural Network (NN) to train two different datasets on Keras with a TensorFlow backend. The parameters of the model are shown in TABLE IV. We then equally allocate data per client, that is, for MNIST and FashionMNIST datasets, each client has about $6,000$ data which is evenly distributed. For the proportion of the malicious clients, we consider different scenarios ranging from 10% to 50%. We compare our scheme with FedSGD and Krum. We set the batch size as $b = 128$. In addition, we observe that the loss function converges after 50 rounds, so we set the training epoch $e = 50$.

B. Experimental Results

We first evaluate the effectiveness of our scheme against poisoning attacks. The results show that our scheme achieves the design goals we discussed in Section IV, including: robustness, privacy, efficiency, accuracy, and reliability. TABLE V and TABLE VI show the training results on these two datasets in the presence of different proportions of malicious clients under targeted and untargeted attacks. And the training processes of 50 iterations on FashionMNIST and MNIST datasets are illustrated in Fig. 5 and Fig. 6, respectively. The results show that we have achieved these five goals.

Specifically, in the presence of different numbers of malicious clients, our scheme still keeps the same accuracy rate as a baseline for different types of poisoning attacks, achieving the goal of robustness and accuracy.

Second, all of our operations, including calculating different scores to clients and model aggregation, are calculated in the ciphertext. No *Solver* and *Verifier* can recover the clients' original plaintext from the ciphertext. Therefore, our scheme meets the clients' requirements for data privacy.

Third, as shown in Fig. 3, as a new and effective encryption scheme, CKKS [24] has its unique advantages in encrypting large-scale floating-point vectors and network models with many parameters. Therefore, compared with other traditional privacy-preserving FL schemes, our scheme achieves the goal of efficiency.

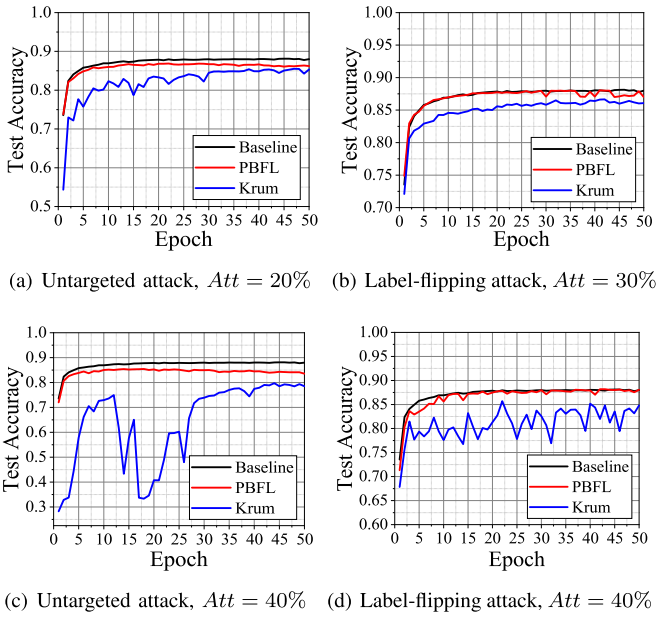
Moreover, the incentive and verification mechanisms ensure that the aggregation is correctly performed, the results of iterations are recorded on blockchain so that they can be traced

TABLE V
TEST ERROR RATES OF PBFL AND KRUM IN THE CASES OF DIFFERENT NUMBERS OF MALICIOUS UNDER ATTACKS (UNTARGET IS UNTARGETED ATTACK AND TARGET IS TARGETED ATTACK) ON MNIST

scheme (attacks)	proportion of malicious clients				
	10%	*20%*	*30%*	*40%*	*50%*
PBFL(Untarget)	0.02	0.02	0.03	0.03	0.04
Krum[14](Untarget)	0.03	0.05	0.06	0.20	0.42
PBFL(Target)	0.02	0.03	0.03	0.03	0.04
Krum[14](Target)	0.03	0.04	0.06	0.16	0.22

TABLE VI
TEST ERROR RATES OF PBFL AND KRUM IN THE CASES OF DIFFERENT NUMBERS OF MALICIOUS UNDER ATTACKS (UNTARGET IS UNTARGETED ATTACK AND TARGET IS TARGETED ATTACK) ON FASHIONMNIST

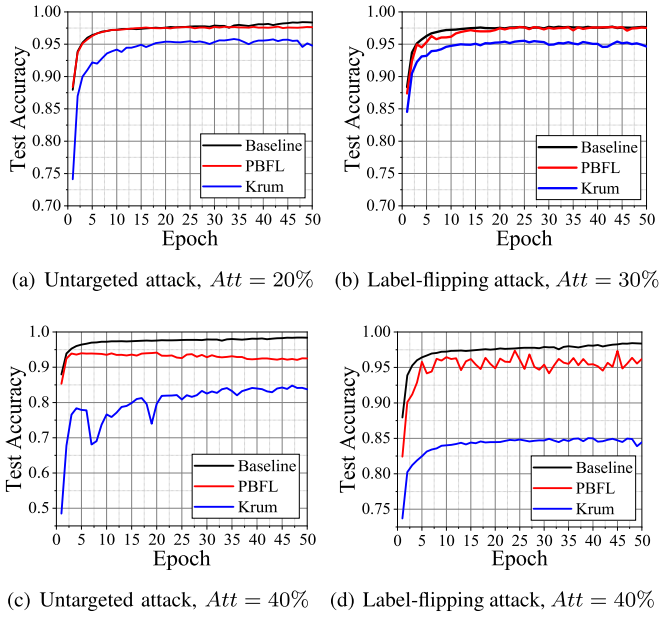
scheme (attacks)	proportion of malicious clients				
	10%	*20%*	*30%*	*40%*	*50%*
PBFL(Untarget)	0.13	0.13	0.14	0.15	0.15
Krum[14](Untarget)	0.15	0.16	0.17	0.21	0.46
PBFL(Target)	0.12	0.12	0.13	0.13	0.14
Krum[14](Target)	0.13	0.13	0.15	0.20	0.28



(a) Untargeted attack, $Att = 20\%$ (b) Label-flipping attack, $Att = 30\%$

(c) Untargeted attack, $Att = 40\%$ (d) Label-flipping attack, $Att = 40\%$

Fig. 5. Test accuracy on Fashion-MNIST, where Att denotes percentage of malicious clients.



(a) Untargeted attack, $Att = 20\%$ (b) Label-flipping attack, $Att = 30\%$

(c) Untargeted attack, $Att = 40\%$ (d) Label-flipping attack, $Att = 40\%$

Fig. 6. Test accuracy on MNIST, where Att denotes percentage of malicious clients.

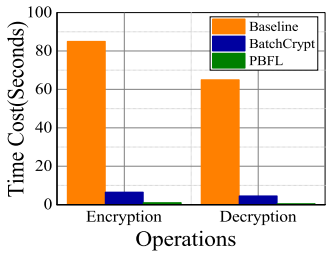


Fig. 7. The cost time of encryption and decryption under BatchCrypt, PBFL and baseline.

back at any time, achieving the goal of reliability and avoiding single point of failure problem and malicious behavior of the server.

In addition, we evaluate the performance of our scheme in on-chain and off-chain phases respectively. In off-chain phase, the encryption operation is the most important factor that affects the efficiency of our scheme. We evaluate PBFL and BatchCrypt [25], note that BatchCrypt is a HE-based federated learning scheme that packs many plaintexts to a ciphertext to improve the efficiency. And we use the Paillier-based federated learning scheme as a baseline. For each client, we use the time required for encryption and decryption of each iteration as a criterion. As shown in Fig. 7, we can see that, compared with baseline and BatchCrypt, PBFL has advantages in encryption and decryption, which greatly improves the efficiency of computation and reduces the cost of computation.

In on-chain phase, uploading data to blockchain consumes gas. Therefore, we use gas cost as a measure of the performance in our scheme. Specifically, the gas cost may be related to the number of clients participating in FL system and the network structure used for training. Therefore, we use different network structures to experiment and test the gas cost in each iteration with a different number of clients.

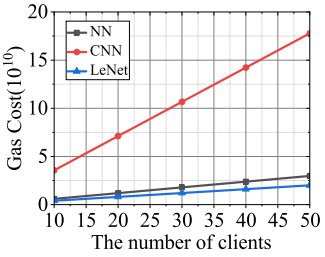


Fig. 8. The total gas cost in each iteration with different numbers of clients and different networks.

TABLE VII

THE NUMBERS OF PARAMETERS IN THE DIFFERENT LAYERS OF THE LENET AND CNN NETWORK, WHERE CONV MEANS CONVOLUTION LAYER AND FC MEANS FULLY CONNECTED LAYER

Nerual Networks	Layer				
	Conv1	Conv2	FC1	FC2	FC3
LeNet	456	2416	48120	10164	850
CNN	832	112704	524800	5130	–

TABLE VIII

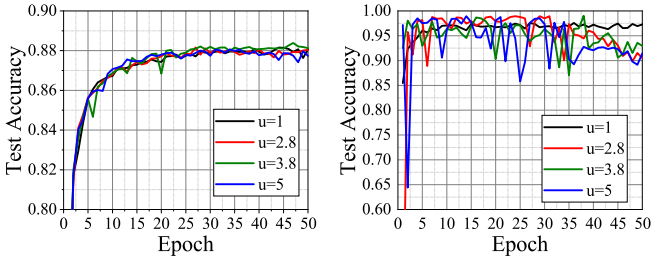
THE TOTAL GAS COST IN THE TRAINING PROCESS WITH DIFFERENT NUMBERS OF CLIENTS USING THE NN NETWORK

Number of clients	10	20	30	40	50
Gas cost(10^{11})	2.97	5.95	8.93	11.91	14.89

One of the NN network is described in TABLE IV, and another two networks are shown in TABLE VII. And the experimental results are illustrated in Fig. 8. Our scheme achieves low gas consumption. Because we leave heavy computation burdens on the servers and only upload the results to blockchain, which greatly reduces the gas consumption. The total gas cost in the whole training process with the NN network is shown in TABLE VIII.

Finally, we explore the impact of the data distribution of the root dataset on the global model. Specifically, We assume that the data on the clients is uniformly distributed. However, the data from the root dataset is not always uniformly distributed due to real-world factors. In our experiment, both MNIST and Fashion-MNIST have ten class labels. When we assign data to the root dataset, the next class always has n' more than the previous class, which is used to simulate the unbalanced distribution of the data from the root dataset. Furthermore, if the first class has a data, then the second class will have $a + n'$ pieces of data, and so on. The tenth class will have $a + 9 * n'$ pieces of data. We use $u = (a + 9 * n')/a$ to indicate the degree of imbalance in the distribution of the root data, where $u = 1$ means that the data is uniformly distributed.

We conduct our experiment based on three malicious clients who initiate the label-flipping attack. Uniformly distribution root data is used as a baseline under the same conditions. We evaluate $u = 2.8$, $u = 3.8$ and $u = 5$, corresponding to the size of the root dataset 190, 235 and 275, respectively. For $u = 2.8$, the class 0 has 10 pieces of data, each subsequent class has 2 more pieces than the previous class. For $u = 3.8$, the amount of data from class 0 to class 10 starts at 10 and increases



(a) Fashion-MNIST (b) MNIST

Fig. 9. The influence of distribution of D_0 on the global model.

by 3 for each class. For $u = 5$, the amount of data from class 0 to class 10 starts at 5 and increases by 5. Our evaluation of Fashion-MNIST and MNIST Datasets is described in Fig. 9.

As illustrated in Fig. 9, tests on different datasets show that distribution differences of data in dataset D_0 do not have catastrophic consequences. In contrast, we can even ignore the effect of data distribution differences on the results, which shows the robustness and practicability of our scheme from another aspect. That is to say, the server does not need to obtain a trusted root dataset containing a large number of samples, and the root dataset D_0 does not have exactly the same data distribution as the client, which reduces the difficulty for the server to obtain the root dataset. According to our evaluation experiments on the root dataset, we have come to the conclusion that in practice, even if the root dataset contains relatively few sample points and the distribution of the root dataset D_0 is different from that of the client, the accuracy of the global model will not be affected.

VIII. Conclusion

In this paper, we propose a new aggregation rule called PBFL to resist poisoning attacks in federated learning and evaluate our scheme using different datasets. The difference of our scheme is that it not only has a trusted root, but also relies on blockchain to promote transparent processes. We also describe the computational details of the scheme in ciphertext environments. Our extensive experiments on different datasets show that our scheme is robust against poisoning attacks, and even a small root dataset can achieve similar results to the non-attack FedSGD scheme. We evaluate our scheme on a balanced distribution of client data. For the cases which the client data is non-IID and very different from the distribution of the root dataset, we leave them for future exploration.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] L. Deng and D. Yu, "Deep learning: Methods and applications," *Found. Trends Signal Process.*, vol. 7, nos. 3–4, pp. 197–387, Jun. 2014.

[3] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1223–1231.

[4] Q. Ho *et al.*, "More effective distributed ML via a stale synchronous parallel parameter server," in *Pro. Int. Conf. Adv. Neural Inf. Process. Syst. (NIPS)*, 2013, pp. 1223–1231.

[5] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol. (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[6] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2017, pp. 1273–1282.

[7] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.)*, 2020, pp. 1605–1622.

[8] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," 2018, *arXiv:1808.04866*.

[9] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *Proc. Eur. Symp. Res. Comput. Secur. (ESORICS)*. Cham, Switzerland: Springer, 2020, pp. 480–501.

[10] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 634–643.

[11] C. Xe, K. Huang, P.-Y. Chen, and B. Li, "DBA: Distributed backdoor attacks against federated learning," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–19.

[12] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2020, pp. 2938–2948.

[13] D. Yin, Y. Chen, K. Ramchandran, and P. L. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 5636–5645.

[14] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 118–128.

[15] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in byzantium," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 3518–3527.

[16] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "FLTrust: Byzantine-robust federated learning via trust bootstrapping," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18, doi: 10.14722/ndss.2021.24434.

[17] Y. Li, C. Chen, N. Liu, H. Huang, Z. Zheng, and Q. Yan, "A blockchain-based decentralized federated learning framework with committee consensus," *IEEE Netw.*, vol. 35, no. 1, pp. 234–241, Jan. 2021.

[18] P. Ramanan and K. Nakayama, "BAFFLE: Blockchain based aggregator free federated learning," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Nov. 2020, pp. 72–81.

[19] J. Eberhardt and S. Tai, "On or off the blockchain? Insights on off-chaining computation and data," in *Proc. Eur. Conf. Service-Oriented Cloud Comput. (ESOCC)*. Cham, Switzerland: Springer, 2017, pp. 3–15.

[20] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*, 1999, pp. 223–238.

[21] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. Annu. Cryptol. Conf. (CRYPTO)*. Cham, Switzerland: Springer, 2013, pp. 75–92.

[22] S. Truex *et al.*, "A hybrid approach to privacy-preserving federated learning," in *Proc. 12th ACM Workshop Artif. Intell. Secur. (AISec)*, 2019, pp. 1–11.

[23] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1333–1345, May 2018.

[24] J. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*. Cham, Switzerland: Springer, 2017, pp. 409–437.

[25] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2020, pp. 493–506.

[26] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1175–1191.

[27] J. Eberhardt and J. Heiss, "Off-chaining models and approaches to off-chain computations," in *Proc. 2nd Workshop Scalable Resilient Infrastruct. Distrib. Ledgers*, Dec. 2018, pp. 7–12.

[28] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. Symp. Found. Comput. Sci. (SFCS)*, Nov. 1982, pp. 160–164.

[29] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, 2018, pp. 1651–1669.

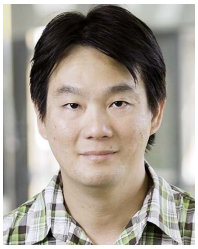
[30] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 1209–1222.

[31] C. Dwork, "Differential privacy: A survey of results," in *Proc. 5th Int. Conf. Theory Appl. Models Comput.*, vol. 4978, 2008, pp. 1–19.
[32] K. Wei *et al.*, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, 2020.
[33] Y. Li, C. Chen, N. Liu, H. Huang, Z. Zheng, and Q. Yan, "A blockchain-based decentralized federated learning framework with committee consensus," *IEEE Netw.*, vol. 35, no. 1, pp. 234–241, Jan. 2021.
[34] J. Cheon, D. Kim, D. Kim, H. Lee, and K. Lee, "Numerical method for comparison on homomorphically encrypted numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, 2019, pp. 415–445.
[35] C. Gentry and D. Boneh, *A Fully Homomorphic Encryption Scheme*, vol. 20. Stanford, CA, USA: Stanford Univ., 2009.
[36] D. Rathee *et al.*, "CrypTFlow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 325–342.
[37] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Commun. Lett.*, vol. 24, no. 6, pp. 1279–1283, Jun. 2020.
[38] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, "Privacy-enhanced federated learning against poisoning adversaries," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4574–4588, 2021.
[39] Y. Dong, X. Chen, K. Li, D. Wang, and S. Zeng, "FLOD: Oblivious defender for private Byzantine-robust federated learning with dishonest-majority," in *Proc. Eur. Symp. Res. Comput. Secur.* Springer, 2021, pp. 497–518.
[40] J. Feng, L. T. Yang, Q. Zhu, and K.-K.-R. Choo, "Privacy-preserving tensor decomposition over encrypted data in a federated cloud environment," *IEEE Trans. Depend. Secure Comput.*, vol. 17, no. 4, pp. 857–868, Jul. 2020.
[41] A. F. M. S. Akhter, M. Ahmed, A. F. M. S. Shah, A. Anwar, and A. Zengin, "A secured privacy-preserving multi-level blockchain framework for cluster based VANET," *Sustainability*, vol. 13, no. 1, p. 400, Jan. 2021.
[42] J. Feng, L. T. Yang, R. Zhang, W. Qiang, and J. Chen, "Privacy preserving high-order Bi-Lanczos in cloud–fog computing for industrial applications," *IEEE Trans. Ind. Informat.*, vol. 18, no. 10, pp. 7009–7018, Oct. 2022, doi: 10.1109/TII.2020.2998086.

Yinbin Miao (Member, IEEE) received the B.E. degree from the Department of Telecommunication Engineering, Jilin University, Changchun, China, in 2011, and the Ph.D. degree from the Department of Telecommunication Engineering, Xidian University, Xi'an, China, in 2016. He is currently a Post-Doctoral Researcher at Nanyang Technological University from September 2018 to September 2019 and a Post-Doctoral Researcher with the City University of Hong Kong from December 2019 to December 2021. He is also an Associate Professor with the Department of Cyber Engineering, Xidian University. His research interests include information security and applied cryptography.

Ziteng Liu received the B.E. degree from the Department of Cyber Engineering, Henan University, Kaifeng, China, in 2021. She is currently pursuing the M.E. degree with the Department of Cyber Engineering, Xidian University, Xi'an, China. Her research interests include information security and applied cryptography.

Hongwei Li (Senior Member, IEEE) received the Ph.D. degree in computer software and theory from the University of Electronic Science and Technology of China, Chengdu, China, in 2008. He has worked as a Post-Doctoral Fellow with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, in 2012. He is currently a Professor with the School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interests include network security, applied cryptography, and trusted computing.



Kim-Kwang Raymond Choo (Senior Member, IEEE) received the Ph.D. degree in information security from the Queensland University of Technology, Australia, in 2006. He is currently holding the Cloud Technology Endowed Professorship at The University of Texas at San Antonio (UTSA). He is also the Founding Co-Editor-in-Chief of ACM Distributed Ledger Technologies: Research & Practice, the Founding Chair of IEEE TEMS Technical Committee on Blockchain and Distributed Ledger Technologies, an ACM Distinguished Speaker and the IEEE Computer Society Distinguished Visitor (2021–2023), and a Web of Science's Highly Cited Researcher (Computer Science–2021, Cross-Field–2020). He was a recipient of the 2019 IEEE Technical Committee on Scalable Computing Award for Excellence in Scalable Computing (Middle Career Researcher).

Robert H. Deng (Fellow, IEEE) has been an AXA Chair Professor in cybersecurity and a Professor in information systems with the School of Information Systems, Singapore Management University, since 2004. His research interests include data security and privacy, multimedia security, networks, and system security. He served/is serving on the editorial boards of many international journals, including IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY and IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING. He has received the Distinguished Paper Award (NDSS 2012), Best Paper Award (CMS 2012), and Best Journal Paper Award (IEEE Communications Society 2017).