PPAPAFL: A Novel Approach to Privacy Protection and Anti-poisoning Attacks in Federated Learning

Xiangquan Chen[1], Chungen Xu[1(✉)], Bennian Dou[1], and Pan Zhang[2]

[1] School of Mathematics and Statistics, Nanjing University of Science and Technology, Nanjing 210094, China
xuchung@njust.edu.cn

[2] School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China

Abstract. In the realm of distributed machine learning, although federated learning has received considerable attention, it still confronts grave challenges such as user privacy leakage and poisoning attacks. Regrettably, the demands for privacy preservation and protection against poisoning attacks are conflicting. Measures for privacy protection generally assure the indistinguishability of local parameter updates, which conversely complicates the strategy of defending against poisoning attacks by making it harder to identify malicious users. To address these issues, we propose a privacy-preserving and anti-poisoning attack federated learning (PPAPAFL) scheme. This scheme employs the CKKS homomorphic encryption technique for gradient packaging encryption, thus ensuring data privacy. Concurrently, our designed robust aggregation algorithm can effectively resist poisoning attacks, guaranteeing the model's integrity and accuracy, and is capable of supporting heterogeneous data in a friendly manner. A plethora of comparative experimental results demonstrate that our scheme can significantly improve the model's accuracy and robustness, drastically reduce the attack success rate, and effectively protect data privacy. In comparison with advanced schemes such as Trum and PEFL, our scheme achieves a 10–50% improvement in model accuracy and reduces the attack success rate to less than 3%.

Keywords: Federated learning · Privacy protection · Homomorphic encryption · Poisoning attacks

1 Introduction

With the advancement of technology and the progress of the era, computing devices have infiltrated every corner of daily life, and the activities of enterprises

Supported by The National Natural Science Foundation of China (No. 62072240) and The Natural Science Foundation of Jiangsu Province Youth Project (No. BK20210330).

J. Liu et al. (Eds.): TridentCom 2023, LNICST 523, pp. 117–138, 2024.
https://doi.org/10.1007/978-3-031-51399-2_7

and individuals are generating massive amounts of data every moment [25]. However, traditional centralized machine learning (ML) methods have encountered challenges in dealing with and utilizing data generated by deployment and application programs that are continuously emerging and widely distributed. Firstly, due to infrastructure constraints, such as limited communication bandwidth and a shortage of storage devices, it takes a significant cost and time to collect these big data [23] into a centralized storage facility. Secondly, private data often contains a lot of sensitive information of individuals or companies [1], such as facial images, identity information, medical records, etc., causing many users or companies to resist sharing their sensitive data with third parties. As society becomes more aware of privacy protection, legal constraints, such as the General Data Protection Regulation (GDPR) in Europe, are gradually being implemented, posing new challenges to data collection and compilation methods.

Federated learning(FL) [15,17], as an emerging paradigm of distributed learning, mainly targets dispersed data. It involves multiple clients (such as smartphones, Internet of Things devices [24], medical institutions, etc.) and a service provider (such as Google, Tencent, etc.). In simple terms, the workflow of FL primarily comprises three iterative steps. Firstly, the server of the service provider disseminates the current global model to clients or a selected subset of them. Secondly, each selected client conducts model training using its local training data and returns the local model update to the server. Finally, the server integrates all local model updates into a global model update according to a pre-established aggregation rule and uses this to update the global model. This method allows many participants to construct a shared machine learning model without exposing their private training data. Moreover, it takes advantage of the diversity of the participants' local datasets to improve the model's performance. Based on these advantages, FL has seen wide applications in recent years, showing exemplary performance in areas such as autonomous driving [9], drug development [7], input method word prediction [12], and recommendation systems [26].

Despite the fact that FL is an ideal distributed learning model, it also faces significant challenges. Firstly, from the perspective of data privacy, existing research shows that the parameters generated during the training process can lead to privacy leaks [11,20,21]. For example, the server may use the received parameters to recover sensitive information about the client. In addition, external malicious attackers could also infer sensitive information of training data by listening to communication channels or observing shared parameter updates. For instance, Membership Inference attacks [21] can use a trained model to determine whether a sample comes from a specific training set, which could lead to the leakage of private information; Model Inversion attacks [11] could generate prototypes of the original training set that should be private by training a model called a Generative Adversarial Network (GAN). Secondly, FL also faces the threat of poisoning attacks [2,10]. As shown in Fig. 1, since the server cannot access the client's dataset and the joint training process, a malicious client could potentially disrupt the local model update by adding poisoned data to the training data (i.e., data poisoning attack) or directly tampering with the model

update (i.e., model poisoning attack), thereby posing a threat to the robustness of the model. In conclusion, a trustworthy FL system must satisfy two requirements: (1) it can protect user privacy data from being leaked, ensuring data privacy; (2) it can effectively resist poisoning attacks, ensuring system security and robustness.

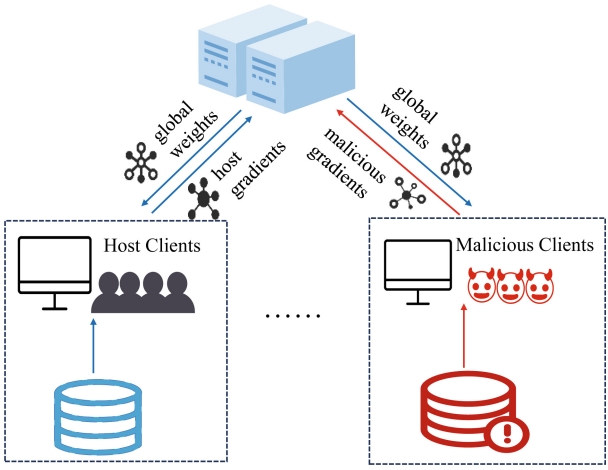


Fig. 1. Poisoning attacks in federal learning

Facing the problem of simultaneously achieving privacy protection and defending against poisoning attacks in FL, current research still faces the following two major challenges: On the one hand, there is an inherent conflict between existing privacy-protecting federated learning (PPFL) methods and measures against poisoning attacks. Mainstream PPFL solutions, such as homomorphic encryption and differential privacy, protect privacy by ensuring that gradients are indistinguishable, which undoubtedly provides superior cover for uploaded malicious gradients. However, strategies to defend against poisoning attacks need to access and analyze uploaded gradients, which undoubtedly leads to leakage of gradients. Therefore, it is difficult for both of these to be satisfied simultaneously. On the other hand, existing defense solutions do not adequately support heterogeneous data. FL usually needs to be trained on various heterogeneous data. Normal gradients trained on IID data usually follow similar distributions, therefore, toxic gradients with different distributions are easy to detect. However, the distribution of benign gradients trained on non-IID data may also be drastically different, making it difficult for existing defense strategies to detect toxic gradients. Furthermore, since the distribution of local data and local gradients is unknown in PPFL strategies, it is particularly challenging to design a privacy-protecting strategy to defend against poisoning attacks in PPFL with heterogeneous data.

Recently, a method called FLTrust has been proposed by Cao et al. [6], which primarily involves the collection of a pristine root dataset. This process identifies poisoned gradients by measuring the cosine similarity between each local gradient and the benign gradient derived from training on the clean data. However, it contradicts the initial principle of FL, which emphasizes data should not leave its original domain, and it is unable to meet the requirement for privacy preservation. Li et al. [16] have put forth a defense against poisoning in FL, a method named LoMar. This approach identifies outliers through kernel density estimation. Regrettably, it functions predominantly under the assumption that most participants are honest.

To solve the aforementioned challenges, we propose PPAPAFL, an innovative defensive strategy aimed at resisting poisoning attacks and protecting privacy. We first effectively encrypt the gradients using the CKKS homomorphic encryption scheme [8], ensuring the security of user data. The advantage of CKKS is that it not only supports packed encryption of real vector gradients, but is also superior in security and efficiency to other encryption strategies. Next, we propose using the Pearson coefficient for statistical analysis of gradients, which can better assess the maliciousness of users than methods such as Euclidean distance and cosine similarity. Finally, we construct a new robust aggregation mechanism to minimize the impact of toxic gradients, thereby ensuring the accuracy of the model. In summary, our main contributions include:

- We have designed a novel FL defensive strategy, called PPAPAFL, which uses CKKS homomorphic encryption as the core technology for PPFL. By performing statistical analysis on each gradient and scoring their maliciousness, it achieves the identification of malicious gradients, effectively solving the privacy protection and poisoning attack problems in FL.
- We propose an innovative robust aggregation mechanism, which significantly improves the robustness of the final model by adaptively reducing the contribution weight of malicious gradients.
- We conduct in-depth theoretical analysis of PPAPAFL and conduct extensive experiments on real data. We also compare our strategy with existing methods (such as Krum, PEFL, FoolsGold, etc.). Experimental results show that our scheme can effectively resist poisoning attacks and maintain high model accuracy.

2 Preliminaries

2.1 Federal Learning

FL is a distributed machine learning architecture composed of a server and N clients (participants). Each client owns its unique dataset $D_i = \{x_1^i, \cdots, xl_i{}^i\}$, with $|Di| = n_i$. During the t-th iteration of training, a subset of n clients are randomly selected from the N clients. These selected clients run the stochastic gradient descent (SGD) algorithm on their local data subsets to calculate local model updates g_i^{t+1}. After obtaining the local updates, the clients upload g_i^{t+1}

to the server. The server then aggregates all the local gradients uploaded by the clients using the formula $w^{t+1} = w^t + \sum i \in n\alpha_i g_i^{t+1}$ to obtain the updated global model parameters. Here, w^t represents the global model parameters after the t iteration, $n = \sum i = 1^N n_i$, and $\frac{n_i}{n} = \alpha_i$.

FEDDECORR: Given the poor support of FL for heterogeneous data, Shi et al. [22] found that data heterogeneity might lead to severe dimension collapse in the global model, causing features to tend towards a low-dimensional space. Therefore, they proposed the FEDDECORR method. This method introduces a regularization term during local training to reduce the correlation between features of different dimensions, thereby alleviating the problem of dimension collapse. The specific expression of this regularization term is as follows:

$$L_{\text{FedDecorr}}\ (w, X) = \frac{1}{d^2}|K|_F^2 \tag{1}$$

where w represents the model parameters, K denotes the correlation matrix, and d refers to the dimension of the matrix K. Hence, the goal of local updates is to minimize the loss function:

$$\min_w \ell(w, X, y) + \beta L\text{FedDecorr}\ (w, X) \tag{2}$$

where ℓ represents the cross-entropy loss, and β is the coefficient of the regularization term.

As the FEDDECORR model can handle heterogeneous data friendly and its additional computational overhead is small, this study adopts the FEDDECORR algorithm for local training. The pseudocode of this algorithm is shown in Algorithm 1.

2.2 Homomomorphic Encryption

Homomorphic encryption is a cryptographical approach allowing computations to be executed on encrypted data, obviating the necessity to decrypt the data beforehand. The outcome of computations on encrypted data, once decrypted, is identical to that obtained directly from plaintext calculations. Specifically, homomorphic encryption enables both addition and multiplication operations to be performed on encrypted data while retaining its encrypted state, rendering decryption unnecessary. Given an encryption algorithm E, a key k, a function f, and an input x, we designate E as homomorphic encryption if the following relationship is satisfied:

$$E_k\left(f\left(x_1, \ldots, x_n\right)\right) = f\left(E_k\left(x_1\right), \ldots, E_k\left(x_n\right)\right) \tag{3}$$

If Eq. (3) applies only to either addition or multiplication, we term such an encryption scheme as partial homomorphic encryption, exemplified by RSA or Paillier schemes. Conversely, if it applies to both, we term it as fully homomorphic encryption, as in the BGV or CKKS schemes. Primary application scenarios of homomorphic encryption include secure computation, such as data

Algorithm 1. FEDDECORR With Federated Average

Let B be the minibatch size for local model training, n_i be the size of dataset in clients d_i, $n = \sum_{i=1}^{N} n_i,$, and η be the learning rate
The server initializes w_0
//server
for each round $t = 1, 2, 3 \ldots$ **do**
  $S_t \leftarrow$ (server randomly selects a set of clients d_i sized of N)
  for each remote client $d_i \in S_t$ in parallel **do**
    $w_{t+1}^i \leftarrow$ ClientUpdate (d_i, w_t)
  end for
  $w_{t+1} \leftarrow \frac{1}{N} \sum_{i=1}^{N} \frac{n_i}{n} w_{t+1}^i$
  if satisfy termination condition **then**
    break
  end if
end for
//client
ClientUpdate $\left(d_i, w_t^i\right)$.
$S \leftarrow$ (select batches sized of B from local dataset)
//we use FEDDECORR to iterate model
for local epochs $l = 1, \ldots, L$ **do**
  for batch $b \in S$ **do**
    //∇ loss is the gradients of loss function.
    $loss = \ell(w, X, y) + \beta L_{\text{FedDecorr}}\ (w, X)$
    $w_{t+1}^i \leftarrow w_t^i - \eta \nabla loss$
  end for
end for
return w_{t+1}^i to server

sharing and analysis in FL, as it facilitates computation and analysis while preserving data privacy. Homomorphic encryption has seen widespread adoption in sectors like finance, healthcare, and government to ensure the security and confidentiality of private data.

CKKS: The CKKS scheme is a fully homomorphic encryption method, an acronym derived from the names of its developers - Cheon, Kim, Kim, and Song. This scheme enables the encryption of real or complex vector values, facilitating homomorphic addition and multiplication operations, all the while maintaining good security and computational efficiency. Owing to these features, it has found extensive usage in privacy-preserving data analysis and machine learning. Furthermore, as a lattice [19]-based encryption technique, it presents a novel resilience against quantum attacks. Given the CKKS' exceptional capability to support floating-point numbers and its high computational efficiency, it has been chosen to encrypt local updates uploaded by our clients.

2.3 Poisoning Attacks

In a poisoning attack scenario, as outlined by Bhagoji et al. [3], the attacker's primary objective is to manipulate the model into generating incorrect predictions, thereby compromising its robustness. Poisoning attacks can be categorized into two broad types: data poisoning attacks and model poisoning attacks.

Data poisoning attacks involve the incorporation of malicious or tampered data into the training dataset, thereby causing the trained model to perform in a manner that serves the attacker's objectives. This could include disrupting the functioning of the model or manipulating its outcomes. A typical strategy within data poisoning attacks is label flipping attacks [4], where labels of regular data are reversed to a target category, causing misclassification by the model. For instance, an attacker might corrupt the real label "1" to "9", causing the model to misclassify digit "1" as digit "9". Another tactic is the backdoor attack [2], which involves modifying specific features or small areas of the original training data to implant certain backdoor triggers. In this case, the model operates normally when dealing with clean, trigger-free data. However, it changes the predicted target category whenever a trigger is encountered, for example, a specific mark on an image. The aforementioned two types of data poisoning attacks are shown in Figure 2.

Contrary to data poisoning attacks, model poisoning attacks compromise the integrity of the model by altering or replacing its parameters. A classic example of a model poisoning attack is the Byzantine attack [5,27], where malicious gradients are uploaded by attackers to the server, leading to a functional breakdown of the global model.

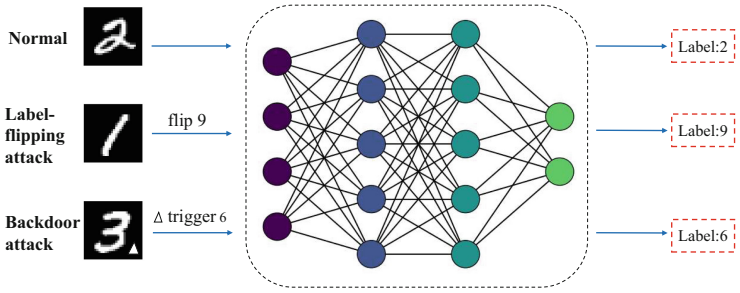


Fig. 2. Label flipping and backdoor attack

2.4 Inference Attacks

Inference attacks [18] involve attackers acquiring certain information through various means (such as eavesdropping or stealing) and using this information to infer details they are interested in. A learning model essentially provides a

high-level statistical representation of its training dataset, enabling attackers to infer a wealth of private information, such as class representatives, membership, and the properties of training data subsets, through gradient inference attacks.

Inference attacks include attribute inference attacks [20], model inversion attacks [14], and membership inference attacks [13]. All these attack methods could potentially lead to the exposure of private data, thereby undermining the objective of FL, i.e., the protection of client-side privacy data.

3 Problem Statement

3.1 System Model

As illustrated in Fig. 3, our system comprises four primary entities:

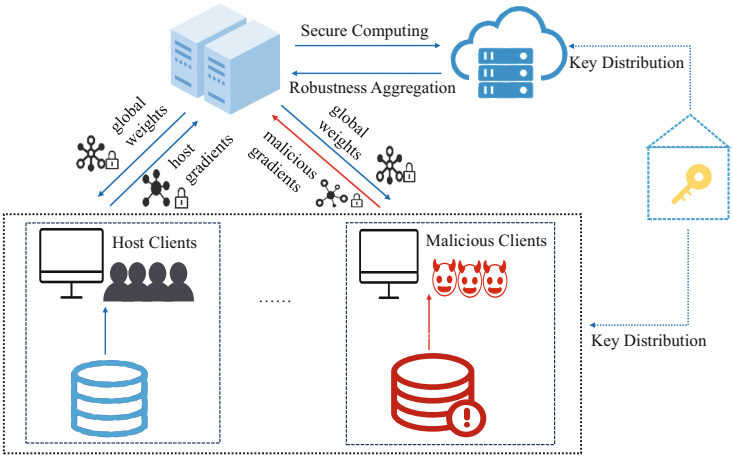


Fig. 3. System Framework

- Key Generation Center (KGC): The KGC is a fully trusted entity, primarily responsible for generating and distributing public keys and private keys. In our scheme, the KGC is in charge of generating public keys (pk) and private keys (sk) for CKKS homomorphic encryption. The public key pk is made public, while the private key sk is securely distributed to the client and the cloud platform.
- Clients(C_i): each client (C_i) possesses its training data. This data is used for model training on the client's local device and the gradient updates produced from this training are encrypted and sent to the server. We assume that the data each client possesses is non-IID.
- Server Provider (SP): The SP is a semi-honest entity that receives all gradient updates from clients and aggregates them into a global optimization model. Additionally, we assume that the SP does not collude with C_i.

- Cloud Platform (CP): The cloud platform assists the SP with computations, helps identify and eliminate malicious clients, thereby defending against poisoning attacks and ensuring the accuracy of model training. We assume that the CP and SP do not collude.

3.2 Threat Model

In this paper, we primarily consider threats arising from poisoning attacks and inference attacks. For poisoning attacks, we assume that the malicious clients are controlled by sybils. They affect model performance by modifying training data (such as label flipping attacks, backdoor attacks) or altering model parameters. For inference attacks, honest but curious servers and other adversaries can launch inference attacks based on the gradients uploaded by clients. This can recover client's private information through the clients' parameter updates.

Table 1. Symbol Description

Symbol	Description
pk	public key
sk	private key
$[[x]]$	CKKS-encrypted ciphertext x
α_i	the user's reputation score
ρ	Pearson correlation coefficient
η	learning rate
γ_i	randomized cryptographic gradient
g_i	user local gradient
W	global model parameters
β	dirichlet distribution
ξ	user Poisoning ratio

3.3 Design Goals

- Privacy: Inference attacks can lead to the leakage of data owners' private information. To protect the data privacy of clients, we need to encrypt the uploaded gradient updates, preventing the unauthorized acquisition of sensitive information during transmission.
- Robustness: Malicious attackers may submit malicious gradients through poisoning attacks, which could cause model performance degradation or even incorrect classification results. To maintain the model's robustness, we need to design a secure aggregation scheme that can identify and eliminate or reduce the impact of poisoned gradients.

4 Our PPAPAFL

In this section, we first provide an overview of our proposed scheme. Then we will delve into the construction of the scheme and its role in enhancing security and privacy protection. For reference, we also list the symbols that appear in this paper and their corresponding descriptions, as shown in Table 1.

4.1 Overview

In our PPAPAFL scheme, we employ the FEDDECORR method for users to conduct local training, effectively addressing the model dimension collapse issue caused by the data heterogeneity among different users. Traditional PPFL schemes, whether for their inadequate support for floating-point gradient or potential model accuracy degradation, present certain issues. To tackle these two problems, we opt to use CKKS technology to encrypt local updates, significantly reducing the communication overhead increase generated by encryption, thereby enhancing efficiency. When considering poisoning attacks, we understand that attackers might manipulate malicious clients, changing the local model update direction, causing global model update direction to be the opposite of the expected, or attackers might magnify the scale of local model updates to dominate the aggregation of global model updates. In order to adeptly address these two issues, the present study selects the Pearson correlation coefficient as the evaluative instrument, owing to its demonstrably superior capability to precisely evaluate the correlation among gradients and to astutely identify anomalous values, especially when contrasted with alternate approaches such as Euclidean distance and cosine similarity. So we use the Pearson correlation coefficient to calculate the similarity of gradient change trends among users, and assign a reputation score to each client based on this correlation coefficient. Finally, we opt to use a secure aggregation scheme to carry out robust aggregation on encrypted gradients according to reputation scores. In a nutshell, the PPAPAFL scheme can effectively support heterogeneous data and comprehensively resolve issues of privacy protection and defense against poisoning attacks.

4.2 PPAPAFL Construction

The PPAPAFL scheme consists of three stages: the system setup stage, the local training stage, and the robust aggregation stage. We will describe these three stages in detail below:

System Setup. Firstly, based on the security parameter λ, the KGC generates the public-private key pair (i.e., pk, sk) required for CKKS homomorphic encryption. The KGC publishes the public key pk to all parties involved, and distributes the private key sk to the data owners C_i and the cloud platform CP via a secure channel. Once the key distribution is completed, the KGC goes offline. Next, the Server Provider (SP) initiates the global parameter W_{init} and distributes the training model to C_i.

Local Training. In this stage, both benign clients (denoted as C) and malicious clients (denoted as C^*) participate in the training process normally. In the t-th training iteration, first, C_i decrypts the global parameter $[[W_t]]$ received from SP to obtain W_t, then carries out local training using FEDDECORR method, which is capable of preventing dimensional collapse of the local model induced by heterogeneous data, thereby obtaining the gradient g_i^t. Given that CKKS homomorphic encryption supports floating-point number encryption and can pack multiple plaintexts into a single ciphertext for encryption, we directly encrypt g_i^t using CKKS, obtaining $[[g_i^t]]$, which effectively saves space and ensures gradient security. Finally, $[[g_i^t]]$ is uploaded to SP. The detailed process is as shown in Algorithm 2.

Algorithm 2. Local Training

Input: Local training data $D_i (i \in [1, n])$, global weight W^t
Output: CKKS encrypted gradients $[[g_i^t]]$
if $C_i \notin C^*$ **then**
  //Benign client local training
  C_i uses W^t training on local data, and gets local gradients g_i
  C_i encrypts g_i then becomes $[[g_i]]$
else
  //malicious client poisoning attack
  C_i^* generating gradients g_i^* using various poisoning attack methods
  C_i^* encrypts g_i^* then becomes $[[g_i^*]]$
end if
return encrypted local gradients $[[g_i]]$ or $[[g_i^*]]$

Robustness Aggregation. In the robust aggregation phase, we engage in secure interactions with the Service Provider (SP) and the Cloud Platform (CP), to identify and evaluate potential malicious clients. Subsequently, we allocate lower reputation scores to malicious gradients, and higher scores to benign ones. Ultimately, secure aggregation is conducted according to these reputation scores, blocking various poisoning attacks launched by malicious users and ensuring high model accuracy.

Secure Pearson Correlation Coefficient: Assume we are in the t^{th} iteration, the server SP initially receives the encrypted gradients $[[g_i^t]]$ trained by each client C_i. Subsequently, the server adds $[[g_i^t]]$ to the sum of the gradients from the previous k rounds $\sum_{i=t-k}^{t-1}[[g_i]]$ (we assume SP will retain its encrypted historical gradients and destroy them at the end of training), the result denoted as $[[G_i]] = \sum_{i=t-k}^{t}[[g_i]]$. This step aims to better judge the correlation of each gradient. Then, SP multiplies all $[[G_i]]$ by a random number r_i, and sends the result to the CP. Upon receiving $\gamma_i = r_i * [[G_i]]$, CP first decrypts it using the private key sk to get γ_i. Subsequently, CP calculates the Pearson correlation coefficient of γ_i

with all other γ_j $(i \neq j)$. Finally, CP sends the Pearson correlation coefficients to SP. Algorithm 3 illustrates the detailed steps of the protocol.

Computing Reputation Score: After receiving the Pearson correlation coefficient $\rho_{i,j}$ of each gradient, SP starts to calculate the maximum Pearson correlation coefficient of each C_i with the remaining C_j, denoted as $\alpha_i = \max_j(\rho_{i,j})$. Given the non-IID nature of the data, benign and malicious gradients might exhibit similarities. Therefore, to prevent benign clients from being misjudged as malicious, we propose to calculate the ratio $\alpha_{i,j} = \alpha_i/\alpha_j$, and to find the maximum α_i to reduce false positives. Following this, we implement a 0-1 inversion and normalization of α_i. This step ensures that malicious gradients have lower reputation scores, and all values adhere to a common standard. Finally, we compute

$$\alpha_i \leftarrow \min\left\{1, \max\left\{0, \ln\left(\frac{1+\alpha_i}{1-\alpha_i}\right) - 0.5\right\}\right\} \tag{4}$$

This enhances the divergence of values close to both ends, thus more effectively distinguishing between benign and malicious gradients. This approach aims to reward benign gradients, while penalizing malicious ones. Ultimately, all results are truncated to fall within the 0–1 range, facilitating the participation of reputation scores in global update calculations.

Algorithm 3. SPCC: Secure computation of Pearson correlation coefficient

Input: SP has $[[G_i]]_{i=1}^{i=n}$; CP has private key sk
Output: all Pearson correlation coefficient ρ
1: $\underline{SP}$:
2: // Randomized $[[G_i]]$
3: **for** $i = 1$ to n **do**
4: $\quad \gamma_i = r_i \cdot [[G_i]]$
5: **end for**
6: sends $\{\gamma_i\}_{i=1}^{i=n}$ to CP
7: $\underline{CP}$:
8: // decrypts γ_i with sk and computes the Pearson correlation coefficient $\rho_{i,j}$
9: **for** $i = 1$ to n **do**
10: $\quad d_i = \mathrm{Dec}(sk_c, \gamma_i)$
11: **end for**
12: Calculates $\rho_{i,j} = \frac{\mathrm{Cov}(d_i, d_j)}{\sigma(d_i)\cdot\sigma(d_j)}$
13: Sends $\rho = \{\rho_{i,j}\}_{i=1,j=1}^{i=n,j=n}$ to SP

Secure Aggregation: We redistribute the weight of each gradient based on the reputation score α_i, and then conduct secure gradient aggregation, namely

$$[[W^t]] \leftarrow [[W^{t-1}]] - \eta \sum_{i=1}^{n} \frac{\alpha_i}{\sum_{x=1}^{n} \alpha_x} [[g_i]] \tag{5}$$

Finally, we send the aggregated gradient $[[W^t]]$ to client C_i. Users decrypt this gradient using the private key sk to obtain the global gradient W^t. The client will use this global gradient for local training in the next round.

The process of robustness aggregation is outlined in Algorithm 4. Throughout the aggregation process, the gradients g_i processed by SP are maintained in a homomorphically encrypted state, ensuring gradient confidentiality. We evaluate the potential maliciousness of each client by calculating the correlation of their accumulated gradients and employing reputation scores. As a result, our approach safeguards user privacy, thwarts poisoning attacks by malicious users, and ensures the accuracy of the final model.

Algorithm 4. Robustness Aggregation

Input: The CKKS encrypted local gradients $\{[[g_1]], \ldots, [[g_n]]\}$
Output: Aggregated encryption weights $[[W^t]]$
// Secure Pearson Correlation Coefficient
$\rho \leftarrow \text{SPCC}([[G_i]]_{i=1}^{i=n})$
// Computing Reputation score
for $i = 1$ to n **do**
  $\alpha_i = max_j(\rho_{i,j})$
end for
for $i = 1$ to n **do**
  for $j = 1$ to n **do**
    if $\alpha_i < \alpha_j$ **then**
      $\alpha_i = 1 - max_j(\frac{\alpha_i}{\alpha_j})$
    end if
  end for
end for
$\alpha_i = \min\left\{1, \max\left\{0, \ln\left(\frac{1+\alpha_i}{1-\alpha_i}\right) - 0.5\right\}\right\}$
// Secure Aggregation
$[[W^t]] \leftarrow [[W^{t-1}]] - \eta \sum_{i=1}^{n} \frac{\alpha_i}{\sum_{x=1}^{n} \alpha_x} [[g_i]]$
return the encryption weight $[[W^t]]$

Correctness Analysis. Since the noise of CKKS decryption is negligible, it does not affect the correctness of the algorithm. We mainly discuss the correctness of the SPCC protocol, judging whether the Pearson correlation coefficient between the gradients can be obtained in the blurred gradients in the SPCC, to ensure the correct identification of malicious gradients.

Proposition 1 (Correctness). *In the SPCC protocol, for a given randomized gradient, the Pearson correlation coefficient between the real gradients uploaded by the client can be correctly calculated.*

Proof. To prove the correctness of the protocol, it is equivalent to proving that $\rho_{G_i,G_j} = \rho_{d_i,d_j}$. First, based on the homomorphism of CKKS homomorphic

encryption, we can get $d_i = r_i G_j$ in the SPCC protocol. We know that covariance and standard deviation have the following properties:

$$\begin{aligned} Cov(d_i, d_j) &= E[(d_i - E(d_i))(d_j - E(d_j))] \\ &= E(d_i d_j) - E(d_i)E(d_j) \\ \sigma(d_i)^2 &= E\left[\left(d_i - E(d_i)^2\right)\right] = E\left[d_i^2\right] - E^2(d_i) \end{aligned} \tag{6}$$

Where $E(d_i)$ represents the expectation, according to the above formula, we can derive the equivalent formula for the Pearson correlation coefficient:

$$\rho_{d_i,d_j} = \frac{E(d_i d_j) - E(d_i)E(d_j)}{\sqrt{E\left(d_i^2\right) - (E(d_i))^2}\sqrt{E\left(d_j^2\right) - (E(d_j))^2}} \tag{7}$$

According to the mathematical property of expectation E, we know that $E(d_i) = r_i E(G_i)$, from which we can infer the following conclusion:

$$\begin{aligned} \rho_{d_i,d_j} &= \frac{E(d_i d_j) - E(d_i)E(d_j)}{\sqrt{E(d_i^2) - (E(d_i))^2}\sqrt{E(d_j^2) - (E(d_j))^2}} \\ &= \frac{r_i r_j E(G_i G_j) - r_i r_j E(G_i)E(G_j)}{r_i\sqrt{E(G_i^2) - (E(G_i))^2} r_j \sqrt{E(G_j^2) - (E(G_j))^2}} \\ &= \frac{E(G_i G_j) - E(G_i)E(G_j)}{\sqrt{E(G_i^2) - (E(G_i))^2}\sqrt{E(G_j^2) - (E(G_j))^2}} \\ &= \rho_{G_i,G_j} \end{aligned} \tag{8}$$

In conclusion, we have proven that the Pearson correlation coefficient can also be correctly calculated for randomized gradients.

5 Experimental Evaluation

In this section, we comprehensively evaluate the PPAPAFL framework using real-world data and conduct experiments in a PyTorch environment on Ubuntu system. Our server is equipped with an Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60 GHz and has 256 GB of memory.

5.1 Experimental Settings

Experimental Data. We conducted experiments on the MNIST and Amazon datasets to evaluate the performance of PPAPAFL. For MNIST, the dataset is already divided into 60,000 training samples and 10,000 testing samples. For Amazon, we randomly split the data into a 70% training set and a 30% testing set.

- MNIST: It is a classic dataset of handwritten digit images, containing a large number of handwritten digit images. Each image is 28×28 pixels in size. The MNIST dataset includes 600,000 training samples, with 784 features, covering 10 different categories.
- Amazon: It is a classic product review text dataset, its feature number is large due to the inclusion of rich product and user information. The Amazon dataset contains 1,500 training samples, has 10,000 features, and is divided into 50 categories.

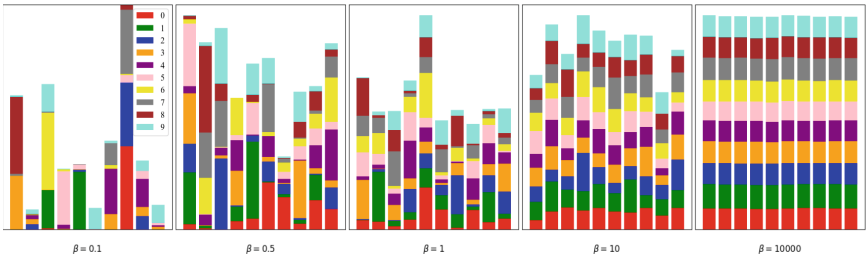


Fig. 4. Dirichlet Distribution of MNIST Data, Divided into 10 Clients and 10 Classes

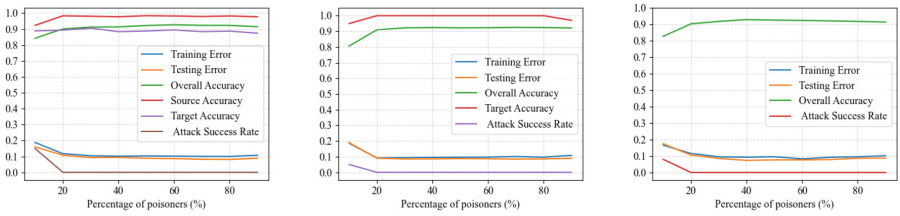
To simulate data heterogeneity among clients, we follow the work of Yurochkin et al. [28] and Shi et al. [22], using probability vectors that follow the Dirichlet distribution to allocate data, as the Dirichlet distribution can effectively capture the heterogeneity and imbalance of data distributions. Each client k obtains instances from the category set $[C] = 1, 2, \ldots, C$ according to the probability vector $p_c = (p_{c,1}, p_{c,2}, \ldots, p_{c,K}) \sim \mathrm{Dir}\, K(\beta)$. Here, pc, k represents the probability of assigning instances of category c to client k. The parameter β controls the degree of data heterogeneity. When β is small, there is a large difference between the data, exhibiting strong heterogeneity. When β tends to infinity, the data becomes more homogeneous. In our experiments, we selected $\beta = \{0.1, 0.5, 1, 10, 10000\}$ to distribute the data to different clients according to these distributions. The distribution is illustrated in Fig. 4.

Model Architecture. Considering that our main focus is to compare the improvement of our method, PPAPAFL, in terms of model accuracy and robustness against various poisoning attacks relative to other methods, rather than absolute accuracy, we first employed a widely used softmax classifier for training, with a batch size of 50. Finally, we trained and evaluated the CNN model on the MNIST data. The client used the cross-entropy loss function and the stochastic gradient descent (SGD) optimizer for model training, with a momentum of 0.9 and a learning rate of 0.001.

Attack Models. Our experiments mainly considered three common poisoning attacks: label flipping attack, backdoor attack, and untargeted model attack. In the label flipping attack, a malicious client modifies the source class to the target class, where class 1 in the MNIST dataset is labeled as class 7, and class 1 in the Amazon dataset is labeled as class 8. In the backdoor attack, we set a malicious client to mark the last pixel point in the dataset as a backdoor mark and label the MNIST dataset as class 7 and the Amazon dataset as class 8. In the untargeted model attack, we allow malicious clients to randomly label the categories in the dataset, leading to incorrect gradients being uploaded, thereby reducing the model accuracy.

5.2 Experimental Results

Firstly, we define the baseline scheme as FL without any defense measures. To compare with our proposed scheme, we selected three defense forms that performed the best, namely Krum, PEFL, and Foolsgold. Next, we will explore the impact of different proportions of poisoners, different numbers of iterations, and different degrees of heterogeneous data on the defense effectiveness of each scheme.

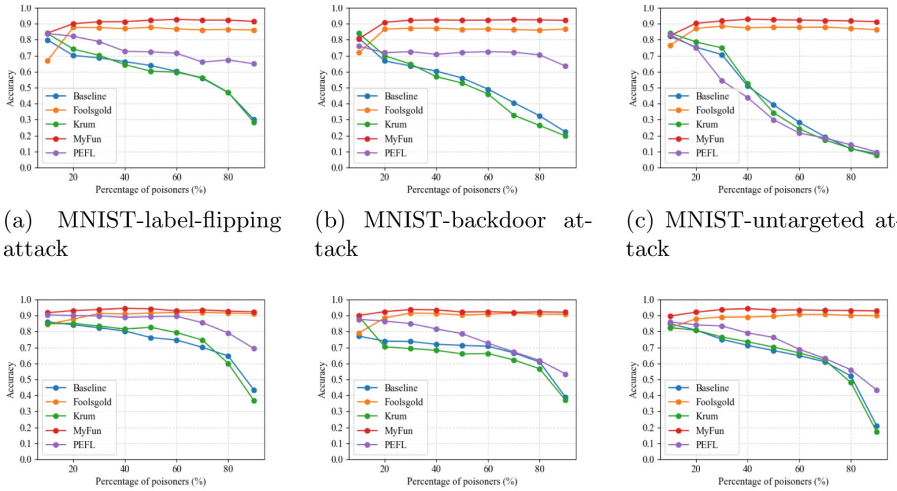


(a) Label-flipping attack (b) Backdoor attack (c) Untargeted attack

Fig. 5. Performance of PPAPAFL

Impact of Different Proportions of Poisoners. We have found that even in traditional federated averaging algorithms, the presence of only a few malicious users can significantly impact the model. As the proportion of malicious users increases, the final model accuracy decreases. In order to explore the impact of the proportion of poisoners on model accuracy and robustness, we conducted experiments where we fixed the number of iterations and used the same heterogeneous data distribution. For the MNIST dataset, we performed 1000 iterations and allocated data with $\beta = 0.1$ to each client. For the Amazon dataset, we conducted 200 iterations and also distributed data with $\beta = 0.1$ to each client.

As shown in Fig. 5, we described the performance of the PPAPAFL scheme in different poisoning attacks when the proportion of malicious users, denoted as ξ, ranged from 10% to 90% on the MNIST dataset. It can be observed that



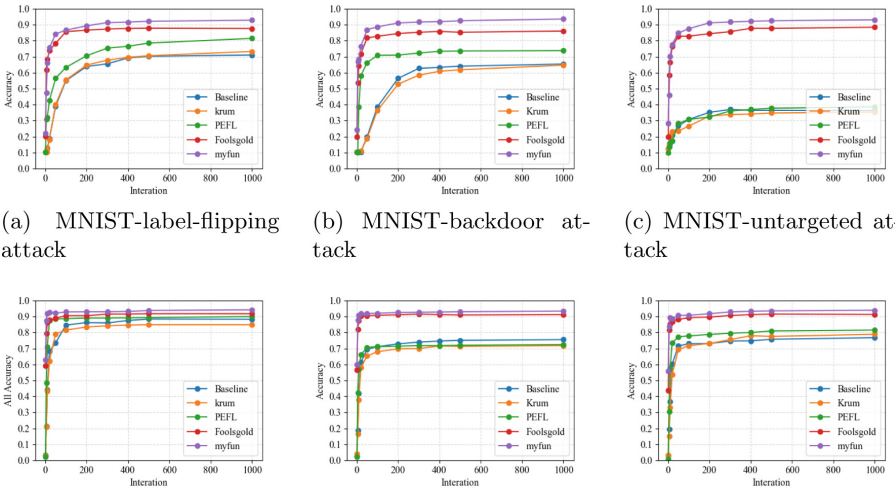
(a) MNIST-label-flipping attack (b) MNIST-backdoor attack (c) MNIST-untargeted attack

(d) Amazon-label-flipping attack (e) Amazon-backdoor attack (f) Amazon-untargeted attack

Fig. 6. Accuracy comparison of poisoners at different proportions

in label flipping attacks, both the training error and testing error of PPAPAFL are exceptionally low, generally within 10%, and the overall accuracy is remarkably high, exceeding 91%. The identification rate for source class 1 and target class 7 is also significantly high, surpassing 90% in almost all instances. The success rate of attacks induced by malicious gradients is extremely low, averaging below 3%, with the majority being zero. When ξ was set to 10%, the slight decrease in performance compared to larger values of ξ was due to the presence of only 10 honest clients, which increased the chances of misclassifying one malicious client as honest, resulting in false negatives. However, this had a minimal impact on the overall model accuracy. The slightly lower performance at $\xi = 90\%$ might be attributed to the similarity gradients between honest clients and a large number of malicious users, resulting in false negatives. However, such cases were rare because they were considered during the design of the scheme. Therefore, the accuracy of PPAPAFL at $\xi = 90\%$ is more than 91%. In the backdoor attack, since there were no source class labels, we only experimented with the recognition accuracy for target class label 7, which was almost perfect at 1. Other metrics were similar to the label-flipping attack. In the untargeted attack, despite its strong destructive nature aimed at reducing model accuracy, the PPAPAFL scheme maintained a remarkably high accuracy, reaching around 90%. Taking into account the aforementioned three types of attacks and various evaluation metrics, we found that the PPAPAFL scheme demonstrated excellent effectiveness in defending against malicious users while maintaining a very high level of model accuracy.

As demonstrated in Fig. 6, it describes the comparison of the overall model accuracy of different defense methods against poisoning attacks under varying numbers of poisoners in the MNIST and Amazon datasets. The comparison reveals that, except for when $\xi = 10\%$, where our scheme is slightly lower than PEFL and Krum, at other ξ values (20%–90%), our scheme's model accuracy significantly surpasses PEFL, Krum, Baseline, and also Foolsgold, achieving an improvement of 5–62%. Notably, the model accuracy of the PEFL, Krum, and Baseline schemes sharply decreases as the number of poisoners increases, especially when it exceeds 40%. This is because these schemes were not designed to handle situations with 50% malicious users, and they are not suitable for handling non-IID data. In contrast, our method can maintain a very high accuracy, essentially reaching above 92% within the range of $\xi = 10\%$–90%. This attests to the effectiveness of our method in enhancing model robustness, and it demonstrates the capability to maintain high accuracy under varying numbers of poisoners.



(a) MNIST-label-flipping attack (b) MNIST-backdoor attack (c) MNIST-untargeted attack

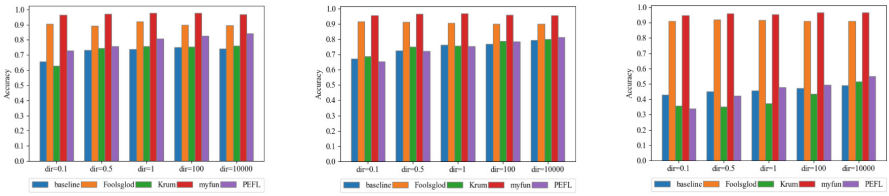
(d) Amazon-label-flipping attack (e) Amazon-backdoor attack (f) Amazon-untargeted attack

Fig. 7. Accuracy comparison of different iterations

Impact of Different Iteration. In FL, model performance typically improves as the number of iterations increases. Initial iterations may only capture some surface features of the data, but with more iterations, the model can better learn the deep features and patterns of the data, fully utilize the local data of each participant for updates and optimization, and achieve global consensus more quickly, thereby improving prediction performance. In FL, having a model capacity that is too small or too large may lead to underfitting or overfitting

problems. However, in this discussion, we ignored these issues and focused on the impact of the number of iterations on model performance. We fixed the proportion of malicious users and the Dirichlet distribution of the data, set $\beta = 0.1$, and $\xi = 50\%$.

As shown in Fig. 7, it compares the changes in model accuracy for different defense methods as the number of iterations increases in the MNIST and Amazon datasets. Based on the results, we can make the following observations and conclusions: In the early iterations, the accuracy of the PPAPAFL scheme rapidly improves as the number of iterations increases. After 50 iterations on the Amazon dataset, the PPAPAFL scheme reached an accuracy of 93% and remained stable. On the MNIST dataset, after 200 iterations, the PPAPAFL scheme reached an accuracy of approximately 92% and remained stable. This indicates that the PPAPAFL scheme has a fast convergence speed and can quickly identify and remove malicious gradients to improve model accuracy during the training aggregation process. The PPAPAFL scheme demonstrates excellent convergence capabilities in the label flipping attack, backdoor attack, and untargeted attack. In comparison, the convergence speed of the PEFL, Krum, and Baseline schemes is slower, and their accuracy is generally lower in most cases due to their inability to defend against poisoning attacks. Particularly, in the untargeted attack on the MNIST dataset, the accuracy of these schemes is only 35%. Although Foolsgold's results are similar to the PPAPAFL scheme, the model accuracy is lower than our scheme because we adopted FEDDECORR for local training and optimized the scheme. In summary, the PPAPAFL scheme can quickly converge as the number of iterations increases and maintain a well-performing model with good robustness. It exhibits excellent performance in various poisoning attack scenarios.



(a) CNN MNIST with the label-flipping attack

(b) CNN MNIST with the backdoor attack

(c) CNN MNIST with the untargeted attack

Fig. 8. Accuracy comparison of different dirichlet distribution

The Impact of Different Dirichlet Distribution Data. In the FL environment, the heterogeneity of data significantly affects the training effectiveness of the model, including its convergence, final performance, and stability. To simulate this impact, we used the Dirichlet distribution to model the data distribution of each client in FL. By adjusting the parameter β of the Dirichlet distribution,

we can change the degree of data heterogeneity, allowing us to observe the impact of different degrees of data heterogeneity on the accuracy and robustness of different defense poisoning attack schemes. We fixed the number of iterations and the Dirichlet distribution of the data, and set $\xi = 50\%$.

As shown in Fig. 8, we tested the cases with β values of $\{0.1, 0.5, 1, 10, 10000\}$ and observed the changes in accuracy for the Baseline, Krum, PEFL, Foolsgold, and PPAPAFL schemes under three poisoning attacks in the MNIST datasets. The results show that under the $\beta = 0.1$ condition, due to the extremely high data heterogeneity, the accuracy of the Baseline, Krum, and PEFL schemes is relatively low, essentially only reaching a precision of 40–80%. As the β value increases, although the accuracy of Baseline, Krum, and PEFL improves to some extent, their accuracy remains relatively low due to their poor defense against malicious users. Foolsgold achieves good accuracy overall, but its accuracy and stability are not as high as PPAPAFL. Our PPAPAFL scheme displays extremely high accuracy for all β values, all reaching over 91%, demonstrating excellent model stability. This indicates that our scheme exhibits good robustness in both non-IID and IID scenarios.

6 Conclusion

we have successfully proposed a novel privacy-preserving and anti-poisoning attack strategy, referred to as PPAPAFL. We employed CKKS homomorphic encryption technique to protect gradients, ensuring the security of users' privacy data. Additionally, we innovatively designed a robust aggregation strategy that effectively identifies and eliminates malicious gradients, significantly enhancing the capability to defend against poisoning attacks. Experimental results have demonstrated that the PPAPAFL scheme maintains high accuracy and robustness for both non-IID and IID data. However, considering that the scheme involves the homomorphic encryption transmission of gradients and the utilization of cloud servers, it incurs certain communication overhead. Therefore, future research efforts should focus on exploring approaches to minimize communication costs while ensuring privacy preservation and attack defense. In summary, our work provides an effective method and theoretical foundation for the study of defense strategies in FL, balancing the demands of user privacy protection and defense against poisoning attacks.

References

1. Abadi, M., et al.: Deep learning with differential privacy. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 308–318 (2016)
2. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: International Conference on Artificial Intelligence and Statistics, pp. 2938–2948. PMLR (2020)

3. Bhagoji, A.N., Chakraborty, S., Mittal, P., Calo, S.: Analyzing federated learning through an adversarial lens. In: International Conference on Machine Learning, pp. 634–643. PMLR (2019)
4. Biggio, B., Nelson, B., Laskov, P.: Poisoning attacks against support vector machines. arXiv preprint arXiv:1206.6389 (2012)
5. Blanchard, P., El Mhamdi, E.M., Guerraoui, R., Stainer, J.: Machine learning with adversaries: Byzantine tolerant gradient descent. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
6. Cao, X., Fang, M., Liu, J., Gong, N.Z.: FLTrust: Byzantine-robust federated learning via trust bootstrapping. arXiv preprint arXiv:2012.13995 (2020)
7. Chen, S., Xue, D., Chuai, G., Yang, Q., Liu, Q.: FL-QSAR: a federated learning-based QSAR prototype for collaborative drug discovery. Bioinformatics **36**(22–23), 5492–5498 (2020)
8. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
9. Du, Z., Wu, C., Yoshinaga, T., Yau, K.L.A., Ji, Y., Li, J.: Federated learning for vehicular internet of things: recent advances and open issues. IEEE Open J. Comput. Soc. **1**, 45–61 (2020)
10. Fang, M., Cao, X., Jia, J., Gong, N.: Local model poisoning attacks to {Byzantine-Robust} federated learning. In: 29th USENIX Security Symposium, USENIX Security 2020, pp. 1605–1622 (2020)
11. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1322–1333 (2015)
12. Hard, A., et al.: Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604 (2018)
13. Hayes, J., Melis, L., Danezis, G., De Cristofaro, E.: LOGAN: membership inference attacks against generative models. arXiv preprint arXiv:1705.07663 (2017)
14. Hitaj, B., Ateniese, G., Perez-Cruz, F.: Deep models under the GAN: information leakage from collaborative deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 603–618 (2017)
15. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
16. Li, X., Qu, Z., Zhao, S., Tang, B., Lu, Z., Liu, Y.: LoMar: a local defense against poisoning attack on federated learning. IEEE Trans. Dependable Secure Comput. **20**, 437–450 (2021)
17. Liu, Y., Xu, L., Yuan, X., Wang, C., Li, B.: The right to be forgotten in federated learning: an efficient realization with rapid retraining. In: IEEE Conference on Computer Communications, IEEE INFOCOM 2022, pp. 1749–1758. IEEE (2022)
18. Lyu, L., Yu, H., Yang, Q.: Threats to federated learning: a survey. arXiv preprint arXiv:2003.02133 (2020)
19. Mei, L., Xu, C., Xu, L., Yu, X., Zuo, C.: Verifiable identity-based encryption with keyword search for IoT from lattice. Comput. Mater. Contin **68**, 2299–2314 (2021)
20. Melis, L., Song, C., De Cristofaro, E., Shmatikov, V.: Exploiting unintended feature leakage in collaborative learning. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 691–706. IEEE (2019)

21. Nasr, M., Shokri, R., Houmansadr, A.: 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), pp. 739–753. IEEE (2019)
22. Shi, Y., Liang, J., Zhang, W., Tan, V.Y., Bai, S.: Towards understanding and mitigating dimensional collapse in heterogeneous federated learning. arXiv preprint arXiv:2210.00226 (2022)
23. Wahab, O.A., Mourad, A., Otrok, H., Taleb, T.: Federated machine learning: survey, multi-level classification, desirable criteria and future directions in communication and networking systems. IEEE Commun. Surv. Tut. **23**(2), 1342–1397 (2021)
24. Xu, L., Xu, C., Liu, Z., Wang, Y., Wang, J., et al.: Enabling comparable search over encrypted data for IoT with privacy-preserving. Comput. Mater. Continua **60**(2), 675–690 (2019)
25. Xu, L., Yuan, X., Zhou, Z., Wang, C., Xu, C.: Towards efficient cryptographic data validation service in edge computing. IEEE Trans. Serv. Comput. **16**, 656–669 (2021)
26. Yang, L., Tan, B., Zheng, V.W., Chen, K., Yang, Q.: Federated recommendation systems. In: Yang, Q., Fan, L., Yu, H. (eds.) Federated Learning. LNCS (LNAI), vol. 12500, pp. 225–239. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63076-8_16
27. Yin, D., Chen, Y., Kannan, R., Bartlett, P.: Byzantine-robust distributed learning: towards optimal statistical rates. In: International Conference on Machine Learning, pp. 5650–5659. PMLR (2018)
28. Yurochkin, M., Agarwal, M., Ghosh, S., Greenewald, K., Hoang, N., Khazaeni, Y.: Bayesian nonparametric federated learning of neural networks. In: International Conference on Machine Learning, pp. 7252–7261. PMLR (2019)