TrustfulFL: Trusted Byzantine-Robust Aggregation in Federated Learning via Martingale Theory.

Jianzhe Chai
School of Information and Software Engineering
University of Electronic Science and Technology of China
Chengdu, China

Jiankun Tang
School of Information and Software Engineering
University of Electronic Science and Technology of China
Chengdu, China

Huayu Xu
School of Information and Software Engineering
University of Electronic Science and Technology of China
Chengdu, China

Ziang Liu
School of Information and Software Engineering
University of Electronic Science and Technology of China
Chengdu, China

***Abstract*—The abstract goes here.**

1. Introduction

Federated Learning has become a crucial branch of traditional Machine Learning owing to its low communication cost and high equipment flexibility. [1] In the federated learning protocols, users upload their local data vectors to the aggregation server. After the aggregation process, updated vectors will be distributed to each user for further training. This iterative and decentralized process allows federated learning to leverage insights from various data sources while preserving data privacy and security. [1]It is particularly suitable for scenarios where centralized data collection is challenging or undesirable, such as edge computing, IoT devices, and privacy-sensitive applications. [2], [3] Compared with traditional centralized learning, federated learning minimizes the risk of leaking private information. At the same time, such a protocol also possesses robustness against training failures.

However, collecting a large amount of private information in federated learning can easily lead to security issues. Currently, most widely used federated learning models are faced with severe privacy protection deficiencies. Data poisoning is the malicious manipulation of local datasets by a participant to compromise the integrity of the global model. A dishonest participant may intentionally include misleading or corrupted data during local model training, aiming to weaken the overall security of the model. Such scenarios may consist of Dishonest clients/servers, Byzantine attacks, and Sybil attacks [4].

Aiming to solve these security issues, we have developed an adaptive method to eliminate attackers in the federated learning system using martingale theory. Axdditionally, we have created a new signature scheme to verify signatures from clients and aggregate gradients from malicious servers in a multi-key environment. This new scheme reduces communication complexity and is more efficient than traditional verification methods. We have also proven the security of this signature scheme under the EU-CMA model.

1.1. System model

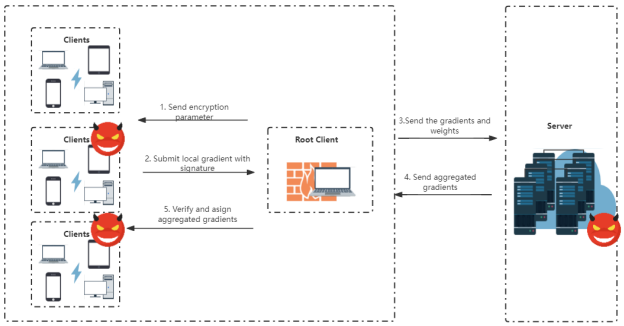

Figure 1. Trustful FL

Our scheme involves two parties: clients and servers. There is a specific client who is always honest, known as the root client. This client holds clean datasets that indicate the direction of the entire training process.

Clients clients, also known as edge devices or local devices, are the individual devices or nodes participating in the federated learning framework. These can be mobile phones, IoT devices, or any device with data to contribute. They train the global model by using their own data. After the aggregation, they receive and update their parameters to the global parameter.

Root client The root client, who is considered completely trustworthy, operates a local model as same as the ordinary client and also collects gradient information from

a subset of other clients. The root client then assigns scores to each collected gradient based on its cosine similarity with the local gradient of the root client's model.How the weight designed will be explained in 3.1. These weights, along with the relevant gradient from each client, are packaged and sent to the server.

After the aggregation, the root client receives aggregated gradient and verifies its correctness.At the same time, root client can verifiy signatures from all clients.

Server The server acts as the central coordinating entity in Trustful-FL scheme. It is responsible for managing the global model. After receiving the packet of weight along with relevant gradient submit by root clients, the server run the aggregation protocol to update the global model and also sent the parameters to the root client.

1.2. Threat Model

We consider similar adversaries as in [5], some users and the server may be malicious and collude with each other to forgery aggregation results. Still, root client is always honest and will not collude with any malicious attacker or participant in our protocol. Therefore, we consider the following threatening adversaries.

N-1 Malicious Clients. The proposed protocol is vulnerable to malicious users who may assist untrusted servers in generating incorrect aggregation results to deceive honest users. These malicious users can even upload harmful gradients, such as Byzantine [6], [7]and poisoned gradients [8].

One Honest Root Client In trustful FL, the root client has to be honest and will not collude with any third party. This is because the root client serves as a trust source of the whole scheme. The root client holds a clean dataset and indicates the direction of the whole training. Furthermore, the root client should also judge which gradients are trusted and mask gradients from clients. It should also verify the aggregated gradient from the malicious server and signatures from clients.

Malicious Server. In the proposed protocol, it is assumed that the aggregate server is malicious. This means that it may attempt to forge aggregation results, and may even collude with a client to obtain their private key in order to create a fake signature for a forged aggregation result, which would pass the verification process..

Third-Party Attackers. The adversaries except the untrusted server and malicious users are called third-party adversaries, which can eavesdrop, analyze, and tamper with the messages transmitted by the public channels.

Under such a threat model, our protocol is Byzantine-robust and verifiable.

1.3. Design Goals

Byzantine-robust In practical scenarios, IoT devices are susceptible to attacks and control, and the distributed nature and single aggregation mechanism of federated learning make it easier for attackers to disrupt the overall learning process. Existing robust aggregation algorithms mostly extend the distributed SGD with robust aggregation rules to ensure that the model obtains robust and effective solutions when subjected to Byzantine attacks. For example, [9] uses the sum of gradient norm distances to select candidate gradients for model parameter updates. [10], on the other hand, start from each dimension of the gradient and select the average of each dimension, excluding the minimum and maximum values, to form a global gradient update. [10] sorts the gradient values for each dimension and takes the median to create a new aggregated gradient. [11]uses a central server as a trust source to evaluate trust score of each client to achieve the byzantine-robust FL, however, This scheme is difficult to converge even when there is no attack. This is because, in the aggregating process, the server will reject all gradients participating in aggregation who have different directions from the server's gradient. However, the direction of the gradient from the server is constantly changing in the convergence stability period. This scheme makes it hard to reach the best point of the global model.

We also use a gradient from a clean dataset to decide which clients are more trustful and should have a higher weight in aggregation in order to adaptive eliminate those clients who have a higher possibility of having Byzantine failures. However, the clean dataset is not from the server as in [11]but from a root client. Another point to make is that our aggregation protocol not only relies on the direction of gradients from clean datasets computed in the current round but also refers to the historical behavior of every client. The direction of convergence is decided by those who have better behavior(have higher trust score) in the previous round. So our scheme has a smoother convergence in the stability period.

Verifiable Aggregation in FL There are already some works about how to verify aggregated model update from server [3], [5], [12].However, these works are all based on Linearly Homomorphic Hash, all of clients share the same the secret key of hash function. As a result, if the malicious server colludes with any client, it will lead to severe private key leakage problems. Malicious servers can easily forgery aggregation results and corresponding hash values. So that such a fake gradient can easily be verified as correct. Furthermore, the Linearly Homomorphic Hash function has a large computation cost, making it unrealistic for practical use. On the other hand, [5] used digital signatures in their scheme, however, they have to verify signatures client by client, which caused great communication cost and computation cost. To solve these problems at the same time, we propose a new signature scheme that can apply

batch verification of all clients' signatures in one round, as well as verify aggregation results from the malicious server in multi-key environment.

Keep Gradient Privet to Malicious Server Considering that malicious server can recover the private training set from the publicly shared gradients [13], we should keep gradients from client inaccessible to malicious server.

2. Related Work

2.1. FL

2.2. Bilinear Pairings

A bilinear pairing [14]can be represented as a map $e : G_1 \times G_2 \to G_I$, where both G_1 and G_2 are multiplicative cyclic groups with the same prime order q. Without loss of generality, we assume that the generators of G_1 and G_2 are g and h, respectively. Informally, a bilinear pairing e has the following properties.

1) **Bilinearity** Given the random numbers $a, b \in Z_q^*$,for any$g_1 \in G_1$and$g_2 \in G_2$ we have $e(g_1^a, g_2^b) = (g_1, g_2)^{ab}$
2) **Computability** $e(g_1, g_2)$ can be computed efficiently for any $g_1 \in G_1$ and $g_2 \in G_2$
3) Non-**degeneracy** $e(g, h) \neq 1$, where g and h are the generator of G_1 and G_2, respectively.

2.3. Martingale theory

Stochastic processes are fundamental in the field of probability theory and statistics. They represent a sequence of random variables that evolve over time, often used to model uncertain or random phenomena. One important class of stochastic processes is martingales. A martingale is a stochastic process where the expected value of future observations, conditional on past information, remains constant over time. This property is expressed mathematically as:

$$\mathbb{E}[X_{t+1} \mid X_0, X_1, \ldots, X_t] = X_t$$
$$\mathbb{E}(|w_r| < \infty)$$

where X_t represents the value of the process at time t, and $\mathbb{E}[\cdot]$denotes the expected value. Martingales have numerous applications in finance, statistics, and various fields where modeling and analyzing random processes is essential. They play a critical role in risk management, option pricing, and the development of statistical methods for inference and prediction

3. Our Trustful FL

3.1. new aggregate rule via Martingale theory

At the beginning of each round ,each client i computes its Model Update gradient g_i using the global model and local datasets. The local update algorithm can be expressed in this way:

Algorithm 1: ModelUpdate

Input: datasets$D$; local learning rate β; number of local iterations R_l; and batch size b.
Output: model update g

$w^0 \leftarrow w$
for *r=1,2,...R* **do**
  Randomly sample a batch D_b from D
  $w^r \leftarrow w^{r-1} - \beta \nabla Loss(D_b; w)$
end
Return $w^r - w$.

After that, we aim to leverage the properties of martingale processes to calculate trust scores for each client during each aggregation round. The trust scores will then be used as weights to apply adaptive adjustments or elimination of the clients.

Let p_0 represent the maximum tolerable probability of negative contribution. Here, n_r is the incentivization rate, which satisfies $n_r > 1$. $\epsilon < 1$ represents the penalty rate.

When it comes to a given client, we establish such rules: "Let w_i represent the trust score of this client in round i. When its contribution is positive in the current round, $w_{i+1} = n_i \cdot w_i$, and when the contribution is negative, $w_{i+1} = \epsilon_i \cdot w_i$.

In our protocol, we use the same way as in [11] to evaluate whether a client made positive contribution to aggregation in a certain round. root client holds a "clean root dataset". root client calculates trusted ModelUpdate parameter g_0 in the clean dataset which works as a trust root of ModelUpdate. An attacker can manipulate the directions of the local model updates on the malicious clients so that the global model update is driven to an arbitrary direction that the attacker desires. During the same training task, if the gradient g_i calculated by client i has more similar direction of g_0, g_i will be more "promising". To evaluate that similarity, we use cosine similarity, a popular metric to measure the deviation between two vectors. In round r, if the cosine similarity of client i local update g_i and trusted Model Update g_0 is larger than a threshold θ, it means that we can trust this update gradient, and we let $TrustedRecord$ $T_i^r = 0$, otherwise we do not trust g_i and let $T_i^r = 1$

$$(\frac{\langle \boldsymbol{g}_i, \boldsymbol{g}_0 \rangle}{\|\boldsymbol{g}_i\| \|\boldsymbol{g}_0\|} > cos(\theta) \,?\, T_i^r = 0 : T_i^r = 1) \tag{1}$$

After evaluating contribution, we try to establish a goal that when a client is not trusted with a probability of p_0, their weight w follows a discrete time martingale. To derive this conclusion, we have

$$E(|w_r| < \infty) \tag{2}$$

$$E(w_{r+1}|w_1, w_2, ...w_r) = w_r \tag{3}$$

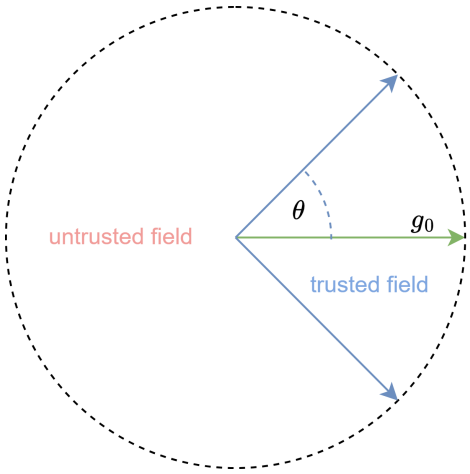

Figure 2. trusted field

Actually, for a certain client, every model update in each round is always from its own dataset and its own dataset is alwsys follow independent and identically distributed (i.i.d) assumption. So that we have

$$E(w_{r+1}|w_1, w_2, ...w_r) = E(w_{r+1}) = w_r \tag{4}$$

$E(w_{i+1})$can also be expressed as:

$$E(w_{r+1}) = (1 - p_0)n_r w_r + p_0 \varepsilon_i w_r \tag{5}$$

which means that when contribution of client is positive in round i, and the probability of it is $(1 - p_0)$, $w_{i+r} = n_r \cdot w_r$, and when the contribution is negative, $w_{r+1} = \epsilon_r \cdot w_r$,and the probability is p_0. Simultaneously solve equation 4 and equation 5, we have

$$(1 - p_0)n_r w_r + p_0 \varepsilon_r w_r = w_r \tag{6}$$

$$\varepsilon_r = \frac{1 - (1 - p_0)n_r}{p_0} \tag{7}$$

When there is a probability of p for a client to be not trusted:

$$E(w_{r+1}) = (1 - p)n_r w_r + p\varepsilon_i w_r \tag{8}$$

Substitute equation 7 we have:

$$E(w_{r+1}) = (1 - p)n_r w_r + p(\frac{1}{1 - (1 - p_0)n_r})w_r \tag{9}$$

$$E(w_{r+1}) = w_r(\frac{(p_0 - p)n_r + p}{p_0}) \tag{10}$$

In our protocol ,we use the frequency of not trusted $f = \frac{\sum_1^r T_i^r}{r}$ after round r to replace the probability of not trusted p. so that we have

$$W_{r+1} = W_r(\frac{(p_0 - f)n_r + f}{p_0}) \tag{11}$$

We also notice that if $f = \frac{p_0 n_r}{n_r - 1}$, the coefficient $\frac{(p_0 - f)n_r + f}{p_0}$ tends to converge to zero and when p is getting larger, $\frac{(p_0 - f)n_r + f}{p_0}$ turns to a negative value. It means that this client has an unacceptable error rate. In such case, we turn $W_r = 0$ to prevent its participation in subsequent aggregations. Our pseudocode for scoring the weights of clients is shown below.

Algorithm 2: Score

Input: client ID i ∈n, $T_i^r \in \{0, 1\}$ means whether client i have positive contribution to aggregation in round r∈ R_g, p_0 means the Maximum tolerable error probability, $1 < n_r < \frac{1}{1-p_0}$is incentive rate, the last round weight of client i W_i^{r-1}

Output: aggregate weight of client i in round r W_i^r

```
f = (∑_1^r T_i^r) / r
if f < p_0 n_r / (n_r − 1) then
    W_i^r = W_{i−1}^r ((p_0 − f) n_r + f) / p_0)
else
    W_{i−1}^r = 0 // It means this client has lost all its credit and is banned to join in the aggregation
end
```

In each round, the root client will deliver $(g_i, W_i)_{i \in n}$ to server for aggregating gradients. However,considering that malicious server can recover the private training set from the publicly shared gradients [13], we should keep gradients from client inaccessible to malicious server. Considering such requirement, masking gradients is necessary. However, we should also make sure that the result of aggregation is still correct. To achieve these goals, we use the mask code to keep the privacy of gradients based on double-masking technology represented by [15]. Our construction of mask code is not same as [15] because: (1) in our protocol we do not need to consider out-line problem because in our scheme, deliver of gradients between root client and server is a one -time communication. (2) The aggregation weights in [15] is all same for every client. But our weight is dynamic not only for different clients but also for different rounds. Our masked gradients can be calculated as follow:

The out put masked gradient $\bar{g}_i$ can be expressed as:

$$\bar{g}_i = g_i + mask_i$$

where

$$mask_i = \sum_{j=1}^{i-1} m_{i,j} \cdot w_j - \sum_{j=i+1}^{n} m_{i,j} \cdot w_j$$

so that when we aggregate $(\bar{g}_1, ..., \bar{g}_n)$ with weight $(w_1, w_2, ..., w_n)$, we have:

$$\sum_{i=1}^{n} w_i \cdot \bar{g}_i$$

Algorithm 3: MASK

Input: the number of client n,wight of each client in round r $(w_1, w_2, ..., w_n)$, gradient of each client in round n $(\bar{\boldsymbol{g}}_1, \bar{\boldsymbol{g}}_2, ...\bar{\boldsymbol{g}}_n)$

Output: masked gradient $(\bar{g_1}, \bar{g_2}, ...\bar{g_i})$

for each two clients, randomly generate a mask code $m_{i,j} = m_{j,i}$ **for** *i=1,2,...n* **do**
  for *j=1,2,...n* **do**
    if *i¿j* **then**
      $mask_i = mask_i + m_{i,j} \cdot w_j$
    else
      if *i¡j* **then**
        $mask_i = mask_i - m_{i,j} \cdot w_j$
      else
        $mask_i = mask_i$
      end
    end
  end
end
for *i=1,2,...n* **do**
  $\bar{g}_i = g_i + mask_i$
end
Return $(\bar{g_1}, \bar{g_2}, ...\bar{g_i})$

$$= \sum_{i=1}^{n} w_i \cdot (g_i + mask_i)$$

$$= \sum_{i=1}^{n} w_i g_i + \sum_{i=1}^{n} w_i mask_i$$

$$= \sum_{i=1}^{n} w_i g_i + \sum_{i=1}^{n} w_i (\sum_{j=1}^{i-1} m_{i,j} \cdot w_j - \sum_{j=i+1}^{n} m_{i,j} \cdot w_j)$$

$$= \sum_{i=1}^{n} w_i g_i + \sum_{i=1}^{n} (\sum_{j=1}^{i-1} m_{i,j} \cdot w_i w_j - \sum_{j=i+1}^{n} m_{i,j} \cdot w_i w_j)$$

$$= \sum_{i=1}^{n} w_i g_i$$

Algorithm 4: Aggregate

Input: the number of client n, wight of each client in round r $(w_1, w_2, ..., w_n)$, gradient of each client in round n $(\bar{\boldsymbol{g}}_1, \bar{\boldsymbol{g}}_2, ...\bar{\boldsymbol{g}}_n)$

Output: aggregate gradient g

$g = \sum_{i=1}^{n} (w_i \cdot \bar{\boldsymbol{g}}_i)$
Return g

3.2. New signature that can verify aggregated weight from malicious server

Our proposed signature scheme can be expressed as follow:

SysGen: The system parameter generation algorithm takes as input a security parameterλ. It chooses a pairing group $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, g, p, e)$,selects a cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}$, and returns the system parameters

$$SP = (\mathbb{PG}, H)$$

.

KeyGen: The key generation algorithm takes as input the system parameters SP, It randomly chooses $\alpha \in \mathbb{Z}_p$ computes$h = g^\alpha$, and returns a public/secret key pair (pk, sk) as follows:

$$pk = h, \quad sk = \alpha$$

Sign: The signing algorithm takes as input a message $m \in \{0,1\}^*$, the secret key sk, and the system parameters SP. It returns the signature σ_m on m as

$$\sigma_m = g^m \cdot H(m)^\alpha$$

Verify: The verification algorithm takes as input a message-signature pair m, σ_m, the public key pk, and the system parameters SP. It accepts the signature if

$$e(\sigma_m,\ g) = e\left(H(m),\ h\right) \cdot e\left(g^m, g\right)$$

The security proof of proposed scheme can be found in section 4.1.

BatchVerification: For n signature and its corresponding public key and massage $(\sigma_i, PK_i, m_i)_{i \in n}$, we can use $BatchVerification$ algorithm to verify them.

The verification algorithm takes as input a set of random mask $(l_1, ...l_n)$, combined message $m = \sum_{i=1}^{n} l_i \cdot m_i$, combined signature $(\sigma = \Pi_{i=1}^{n} {\sigma_i}^{l_i})$ and corresponding public key and hash of massage $(PK_1, ..., PK_n)$, $(H(m_1), ..., H(m_n))$ and the system parameters SP. It accepts all signature and output 1 if

$$e(\sigma, g) == e(g^m, g) \cdot \Pi_{i=1}^{n} e(H(m_i), PK_i)^{l_i}$$

$Correctness$

$$e(\sigma, g) = e(\Pi_{i=1}^{n} {\sigma_i}^{l_i}, g)$$

$$= \Pi_{i=1}^{n} e(\sigma_i, g)^{l_i}$$

$$= e(g^{\sum_{i=1}^{n} l_i m_i}, g) \cdot \Pi_{i=1}^{n} e(H(m_i), PK_i)^{l_i}$$

$$= e(g^m, g) \cdot \Pi_{i=1}^{n} e(H(m_i), PK_i)^{l_i}$$

3.3. Full Trustful FL Algorithm

AS the pseudocode of the whole protocol shown below, The algorithm consists of 5 steps, with the step 0 being executed only in the first round.

In Step 0, the root client generates the Signature system parameter PP. After that, the root client randomly initializes a model. The root client sends PP and begins

sending Weight w_0 to client $i \in n$ using a secure channel.

In Step 1, each client i computes its model update g_i in its own dataset. Then it calculate its Signature σ_i, sends (g_i, σ_i) to root client; at the same time, the root client calculates model update g_0 on its clean dataset.

In Step 2, the root client calculates weights in aggregation for the corresponding client based on its history behavior using $score$ function. After that, the root client will aggregate the signature of each client. Then it will mask all gradients from other clients to ensure privacy. Finally the root client calculates a list with aggregation weights of each client and sends masked gradients and weight list to the server.

In step 3, the server will aggregate masked gradients according to the weight list. After calculation, the server returns the aggregated gradient to all clients.

In step 4, the root client receives gradients from the server, then it starts verifying all signatures from other clients and the correctness of aggregated gradient from the server using $BatchVerication$. If verification passes, all clients go to step 5 to update the global model.

4. Security Analysis

4.1. Security proof of our proposed signature scheme

Theorem 1. Suppose the hash function H is a random oracle. If the CDH problem is complex, the proposed signature scheme is provably secure in the EU-CMA security model with reduction loss L = q_H, where q_H is the number of hash queries to the random oracle.

Proof. Suppose there exists an adversary $\mathscr{A}$ who can$(t, q_s, \mathcal{E})$ -break the signature scheme in the EU-CMA security model. We construct a simulator $\mathscr{B}$ to solve the CDH problem. Given as input a problem instance (g, g^a, g^b) over the pairing group PG, $\mathscr{B}$ controls the random oracle, runs $\mathscr{A}$, and works as follows.

Setup. Let $SP = \mathbb{PG}$ and H be the random oracle controlled by the simulator. $\mathscr{B}$ sets the public key as $h = g^a$, where the secret key α is equivalent to a. The public key is available from the problem instance.

H-Query. The adversary makes hash queries in this phase. Before receiving queries from the adversary, $\mathscr{B}$ randomly chooses an integer $i^\star \in [1, q_H]$, where q_H denotes the number of hash queries to the random oracle. Then, $\mathscr{B}$ prepares a hash list to record all queries and responses as follows, where the hash list is empty at the beginning.

Let the i-th hash query be m_i. If m_i is already in the hash list, $\mathscr{B}$ responds to this query following the hash list. Otherwise, $\mathscr{B}$ randomly chooses w_i from $\mathbb{Z}_p$ and sets $H(m_i)$ as

$$H(m_i) = \begin{cases} g^{b+w_i} & \text{if } i = i^* \\ g^{w_i} & \text{otherwise} \end{cases}$$

The simulator $\mathscr{B}$ responds to this query with $H(m_i)$ and adds $(i, m_i, w_i, H(m_i))$ to the hash list.

Signature Query. The adversary makes signature queries in this phase. For a signature query on m_i, if m_i is the $i^\star$-th queried message in the hash list, abort. Otherwise, we have $H(mi) = g^{w_i}$ $\mathscr{B}$ computes σ_{m_i}as

$$\sigma_{m_i} = g^{m_i} \cdot (g^a)^{w_i}$$

According to the signature definition and simulation, we have

$$\sigma_{m_i} = g^{m_i} \cdot H(m_i)^a = g^{m_i} \cdot (g_i^w)^a = g^{m_i} \cdot (g^a)^{w_i}$$

Therefore, σ_{m_i} is a valid signature of m_i.

Forgery. The adversary returns a forged signature $\sigma_{m^\star}$ on some that has not been queried. If $m^\star$is not the $i^\star$-th queried message in the hash list, abort. Otherwise, the weight is$H(m^*) = g^{b+w_{i^*}}$
According to the signature definition and simulation, we have

$$\sigma_{m^*} = g^{m^\star} H(m^*)^\alpha = g^{m^\star} (g^{b+w_{i^*}})^a = g^{ab+aw_{i^*}+m^\star}.$$

The simulator $\mathscr{B}$ computes

$$\frac{\sigma_{m^*}}{(g^a)^{w_i^*} \cdot g^{m^\star}} = \frac{g^{ab+aw_{i^*}+m^\star}}{g^{aw_{i^*}+m^\star}} = g^{ab}$$

as the solution to the CDH problem instance. This completes the simulation and the solution. The correctness is analyzed as follows.

Indistinguishable simulation. The correctness of the simulation has been explained above. The randomness of the simulation includes all random numbers in the key generation and the responses to hash queries. They are

$$a, w_1, \cdots, w_{i^*-1}, b + w_{i^*}, w_{i^*+1}, \cdots, w_{q_H}.$$

According to the setting of the simulation, where a, b, w_i are randomly chosen, it is easy to see that they are random and independent from the adversary's point of view adversary. Therefore, the simulation is indistinguishable from the actual attack.

Probability of successful simulation and valuable attack. If the simulator successfully guesses $i^\star$, all queried signatures are able to be simulated, and the forged signature is reducible because the message $m_{i^\star}$ cannot be chosen for a signature query. It will be used for the signature forgery. Therefore, the probability of successful simulating a practical attack is $\frac{1}{q_H}$ for q_H queries.

Advantage and time cost. Suppose the adversary breaks the scheme with $(t, q_s, \mathcal{E})$ after making q_H queries to the random oracle. The advantage of solving the CDH problem is therefore $\frac{\epsilon}{q_H}$. Let T_s denote the time cost of the simulation. We have $T_s = O(q_H + q_s)$, mainly dominated by the

Algorithm 5: Trustful FL

Input: n clients with local training datasetsD_i, i = 1, 2,..., n; a server for aggregating gradient ,a trusted third party(root client) with root dataset D_0; global learning rate α; number of global iterations R_g; local learning rate β; number of local iterations R_l; and batch size b;security parameter λ ,the Maximum tolerable error probabilityp_0,$1 < n_r < \frac{1}{1-p_0}$is incentive rate .

Output: Global model w

/* Step 0:root client generate System parameter for every client and randomly initialize a start weight and send to client using secure channel */

$pp = Sysgen(1^\lambda)$

w $\leftarrow$ random initialization;

The root client sends (pp,w) to all clients.;

/* Step I:Training local models and root client model */

for *r = 1, 2, ..., R_g* **do**

// client side do in parallel:

(sk_i, pk_i) =$KeyGen(pp)$ **for** $i = 1, 2, 3...n$ **do**

$g_i = ModelUpdate(w, D_i, b, \beta, R_l)$.;

$\sigma_i = Sign(sk_i, g_i)$;

publish puliic keyPk_i Send(g_i, σ_i)to the root client.

end

// root client side

receive g_i, σ_i from all clients;

$g_0 = ModelUpdate(w, D_0, b, \beta, R_l)$.;

/* Step II:Calculate Weights for aggregating */

// root client side :

for $i = 1, 2, 3...n$ **do**

$(\frac{\langle \boldsymbol{g}_i, \boldsymbol{g}_0 \rangle}{\|\boldsymbol{g}_i\| \|\boldsymbol{g}_0\|} > \theta ? T_i^r = 0 : T_i^r = 1)$;

$W_i^r = Score(i, T_i^r)$

$List^r = (W_1^r, W_2^r, ..., W_n^r)$

end

$\sigma = \Pi_{i=1}^n \sigma_i^{W_i^r}$

$(\bar{g_1}, \bar{g_2}, ...\bar{g_i}) = MASK((g_1, g_2, ..., g_n), List^r)$

Send($List^r, (\bar{g_1}, \bar{g_2}, ...\bar{g_i})$)to server

/* Step III:Server Aggregates gradient */

// Server side :

$g = Aggregate(List^r, (\bar{g_1}, \bar{g_2}, ...\bar{g_i}))$;

return g root client

/* Step IV:root client verify aggregation result */

// root client side :

Π =BatchVerify(List$^r, g, (PK_1, ..., PK_n), (H(m_1), ..., H(m_n)))$;

if $\Pi = 1$ **then**

/* Step V:updating global model */

root client sends g to all clients

// root client and all client sides do in paralle::

$w \leftarrow w + \alpha \cdot g / \sum_{i=1}^n W_i^r$;

else

Return g;

end

end

oracle response and the signature generation. Therefore,$\mathscr{B}$ will solve the CDH problem with $(t + T_s, \epsilon/q_H)$. This completes the proof of the theorem.

5. Evaluate

5.1. Accuracy

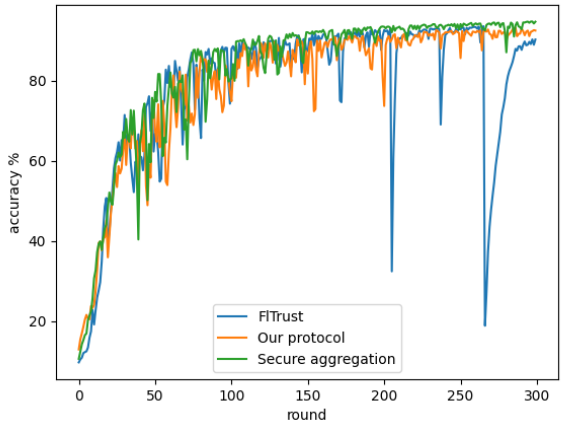

Figure 3. Accuracy without attack

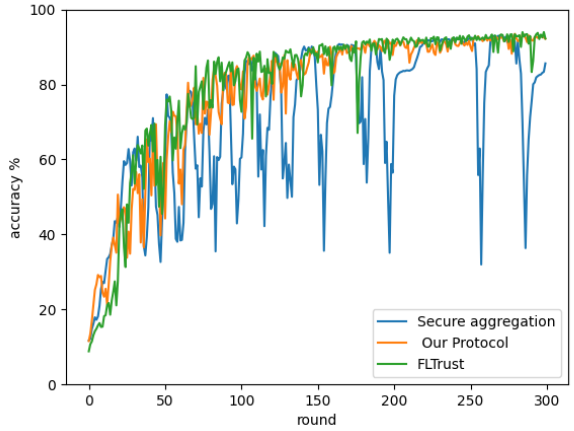

Figure 4. Accuracy with LF attack

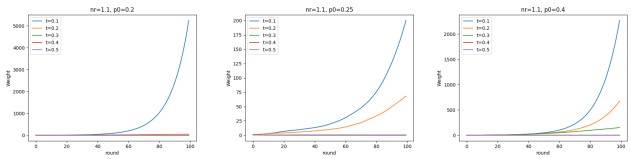

Figure 5. Weight in different t,t means the rate of byzantine gradient

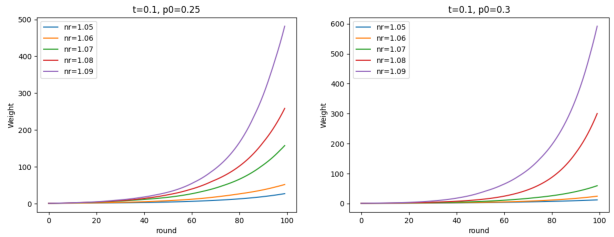

Figure 6. Weight in different n_r

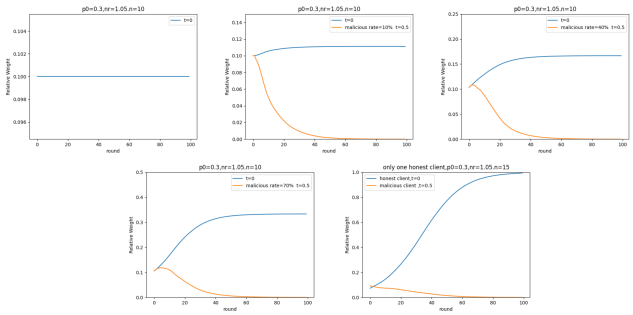

Figure 7. Relative weight of client in aggregation

5.2. Communication cost

5.3. computation cost

5.4. security property

Acknowledgments

The authors would like to thank...

References

[1] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Computers & Industrial Engineering*, vol. 149, p. 106854, 2020.

[2] A. Datta, M. Fredrikson, G. Ko, P. Mardziel, and S. Sen, "Use privacy in data-driven systems: Theory and experiments with machine learnt programs," in *Proceedings of the 2017 ACM SIGSAC conference on Computer and Communications Security*, 2017, pp. 1193–1210.

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

[4] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

[5] Y. Ren, Y. Li, G. Feng, and X. Zhang, "Privacy-enhanced and verification-traceable aggregation for federated learning," *IEEE Internet of Things Journal*, vol. 9, no. 24, pp. 24 933–24 948, 2022.

[6] H. Zhu and Q. Ling, "Bridging differential privacy and byzantine-robustness via model aggregation," *arXiv preprint arXiv:2205.00107*, 2022.

[7] S. P. Karimireddy, L. He, and M. Jaggi, "Learning from history for byzantine robust optimization," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5311–5319.

[8] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[9] Z. Wu, Q. Ling, T. Chen, and G. B. Giannakis, "Federated variance-reduced stochastic gradient descent with robustness to byzantine attacks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 4583–4596, 2020.

[10] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5650–5659.

[11] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," *arXiv preprint arXiv:2012.13995*, 2020.

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

[13] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Advances in neural information processing systems*, vol. 32, 2019.

[14] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual international cryptology conference*. Springer, 2001, pp. 213–229.

[15] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.