TRUSTFULFL

A PREPRINT

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

Elias D. Striatum
Department of Electrical Engineering
Mount-Sheikh University
Santa Narimana, Levand
stariate@ee.mount-sheikh.edu

October 30, 2023

ABSTRACT

1 Introduction

1.1 Contributions

1.2 System model

our scheme includes two party :clients and server. There is a special client who is always honest called Root Client.The root client hold some clean datasets to indicate the direction of whole training.

1.3 Threat Model

we consider the similar adversary as in [1], some users and the server may be malicious and collude with each other to forgery aggregation result, but 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.

malicious aggregate server In the proposed protocol,aggregate server is malicious,which means it will try to forgery aggregation result,it even collude with any client to obtain their private key so that it have ability to forgery a signature of faked aggregation result to pass the verification.

less than N-1 malicious clients In the proposed protocol, malicious users may help untrusted server to generate wrong aggregation result to cheat the honest users. malicious users can even upload wrong gradient, such as Byzantine gradient[2, 3] and poisoned gradient[4] . the honest clients have to be more than one besides root client because root client's gradient only serve as a direction of global model and will not directly participate in aggregation.

Third-Party Attackers The adversaries except untrusted server and malicious users are called the 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 .

1.4 Design Goals

clients behaved better have higher weight in aggregation

verify correctness of aggregation and verify clients' signatures at the same time

keep gradients private from malicious server

2 Related Work

2.1 FL

2.1.1 Byzantine-robust Federated Learning

2.1.2 Verifiable Aggregation in FL

2.2 Bilinear Pairings

2.3 Martingale theory

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 on global model and local datasets.The local update algorithm can be expressed as this:

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, using these trust scores as weights to achieve adaptive adjustments or elimination of various clients

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

In the case of 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 [5] to evaluate whether a client did positive contribution to aggregation in a certain round.ROOT CLIENT holds a "clean root dataset".ROOT CLIENT calculates trusted ModelUpdate g_0 in the clean dataset which works as a trust root of ModelUpdate.An attacker can manipulate the directions of the

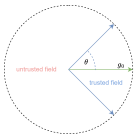

Figure 1: trusted field

local model updates on the malicious clients such that the global model update is driven to an arbitrary direction that the attacker desires.So 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 angle between two vectors.If in round r,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 make $TrustedRecord$ $T_i^r = 0$,otherwise we do not trust g_i and make $T_i^r = 1$

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

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 To derive this conclusion,we have

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

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

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 $\frac{(p_0 - f)n_r + f}{p_0}$ turn to zero and when p is larger,$\frac{(p_0 - f)n_r + f}{p_0}$turn 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

$f = \frac{\sum_1^r T_i^r}{r}$
if $f < \frac{p_0 n_r}{n_r - 1}$ **then**
  $W_i^r = W_{i-1}^r(\frac{(p_0 - f)n_r + f}{p_0})$
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 ,root client will deliver $(g_i, W_i)_{i \in n}$ to server for aggregating gradients.However,considering that malicious server can can recover the private training set from the publicly shared gradients[6],we should keep gradients from client not accessible to malicious server.So that 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 [7].Our construction of mask code is not same as [7] 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 [7] 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:

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 $(\bar{g_1}, \bar{g_2}, ...\bar{g_i})$

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:

$$\begin{aligned}
&\sum_{i=1}^{n} w_i \cdot \bar{g}_i \\
&= \sum_{i=1}^{n} w_i \cdot (g_i + mask_i) \\
&= \sum_{i=1}^{n} w_i g_i + \sum_{i=1}^{n} w_i mask_i \\
&= \sum_{i=1}^{n} w_i g_i + \sum_{i=1}^{n} w_i (\sum_{j=1}^{i-1} m_{i,j} \cdot w_j - \sum_{j=i+1}^{n} m_{i,j} \cdot w_j) \\
&= \sum_{i=1}^{n} w_i g_i + \sum_{i=1}^{n} (\sum_{j=1}^{i-1} m_{i,j} \cdot w_i w_j - \sum_{j=i+1}^{n} m_{i,j} \cdot w_i w_j) \\
&= \sum_{i=1}^{n} w_i g_i
\end{aligned}$$

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 at the same time

There are already some works about how to verify aggregated model update from server[8, 9, 1],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 serious private key leakage problem.Malicious server can easily forgery aggregation result and corresponding hash value.So that such a fake gradient can easily be verified as correct.Furthermore , Linearly Homomorphic Hash function has a great computation cost,which makes it unrealistic for practical use. On the other hand,[1] used digital signature in their scheme ,however,they have to verify signature client by client,which caused great communication cost and computation cost. To solve these problem at the same time,we propose a new signature scheme which can batch verify all clients' signatures in one round ,it can also verify aggregation result from malicious server in multi-Key environment.

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

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

$Correctness$

$$\begin{aligned} e(\sigma, g) &= e(\Pi_{i=1}^{n} {\sigma_i}^{l_i}, g) \\ &= \Pi_{i=1}^{n} e(\sigma_i, g)^{l_i} \\ &= e(g^{\sum_{i=1}^{n} l_i m_i}, g) \cdot \Pi_{i=1}^{n} e(H(m_i), PK_i)^{l_i} \\ &= e(g^m, g) \cdot \Pi_{i=1}^{n} e(H(m_i), PK_i)^{l_i} \end{aligned}$$

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 the Step 0,root client generate Signature system parameter PP . After that, root client randomly initialization a model. root client send PP and begining Weight w_0 to client $i \in n$ using a secure channel.

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

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

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

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

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

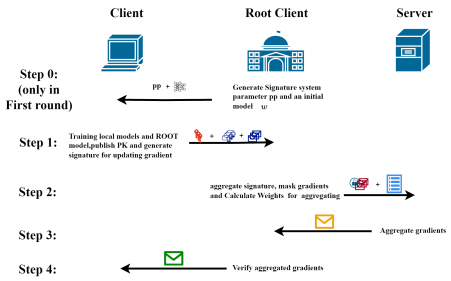

Figure 2: Trustful FL

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 hard, 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, $\sigma_{m_i}$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, we have $H(m^*) = g^{b+w_{i^*}}$
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 point of view of the adversary. Therefore, the simulation is indistinguishable from the real attack

Probability of successful simulation and useful attack If the simulator successfully guesses $i^\star$, all queried signatures are simulatable, and the forged signature is reducible because the message $m_{i^\star}$ cannot be chosen for a signature query, and it will be used for the signature forgery. Therefore, the probability of successful simulation and useful 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 $T_s = O(q_H + q_s)$, which is mainly dominated by the 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

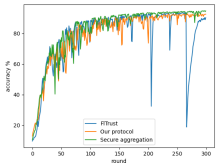

Figure 3: Accuracy without attack

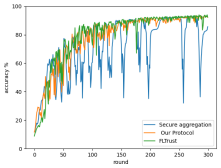

Figure 4: Accuracy with LF attack

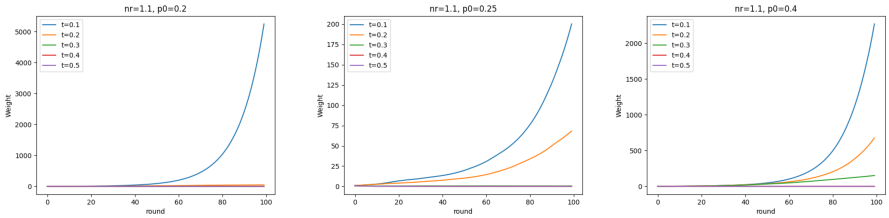

Figure 5: Weight in different t,t means the rate of byzantine gradient

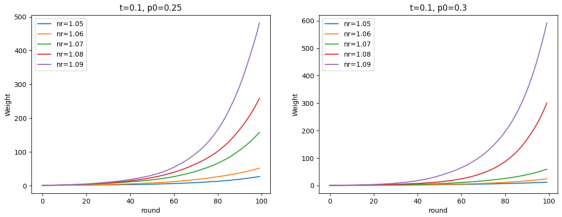

Figure 6: Weight in different n_r

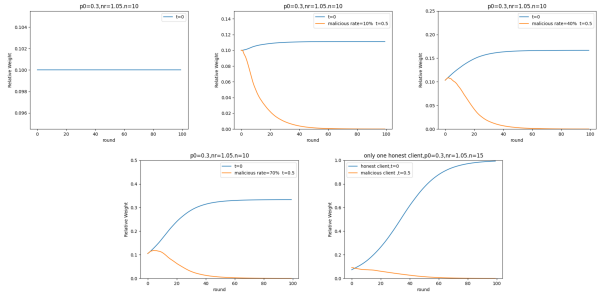

Figure 7: Relative weight of client in aggregation

5.2 Communication cost

5.3 computation cost

5.4 security property

References

[1] Yanli Ren, Yerong Li, Guorui Feng, and Xinpeng Zhang. Privacy-enhanced and verification-traceable aggregation for federated learning. *IEEE Internet of Things Journal*, 9(24):24933–24948, 2022.

[2] Heng Zhu and Qing Ling. Bridging differential privacy and byzantine-robustness via model aggregation. *arXiv preprint arXiv:2205.00107*, 2022.

[3] Sai Praneeth Karimireddy, Lie He, and Martin Jaggi. Learning from history for byzantine robust optimization. In *International Conference on Machine Learning*, pages 5311–5319. PMLR, 2021.

[4] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[5] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. *arXiv preprint arXiv:2012.13995*, 2020.

[6] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019.

[7] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[8] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. Verifynet: Secure and verifiable federated learning. *IEEE Transactions on Information Forensics and Security*, 15:911–926, 2019.

[9] Xiaojie Guo, Zheli Liu, Jin Li, Jiqiang Gao, Boyu Hou, Changyu Dong, and Thar Baker. V eri fl: Communication-efficient and fast verifiable aggregation for federated learning. *IEEE Transactions on Information Forensics and Security*, 16:1736–1751, 2020.