SCIENCE CHINA
Information Sciences

• **Supplementary File** •

Efficient Privacy-Preserving Federated Learning under Dishonest-Majority Setting

Yinbin Miao[1], Da Kuang[1], Xinghua Li[1], Tao Leng[2,3,4*], Ximeng Liu[5] & Jianfeng Ma[1]

[1]*School of Cyber Engineering, Xidian University, Xi'an* 710071*, China;*
[2]*Intelligent Policing Key Laboratory of Sichuan Province, Sichuan Police College, Luzhou* 646000*, China;*
[3]*Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100085*, China;*
[4]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing* 100049*, China;*
[5]*Key Laboratory of Information Security of Network Systems, Fuzhou University, Fuzhou* 350108*, China*

Appendix A Technical

Appendix A.1 Preprocessing Operation

Sample-based Secret Sharing. The method is designed according to the characteristics of computing Hamming distance, which can efficiently compute Hamming distance under ciphertext and also support weighted average without the process of Bit2A. And the process is shown in Algorithm A1. Given n participants $\{P_i | i \in [0, n-1]\}$ and a secret distributor D holding a binary vector $M = (m_0, ..., m_{k-1})_2$ of length k, D first initializes n vectors $[\![m]\!]_i$ of length k with 0, then initializes a random seed locally to generate a random sequence $Ind = (Ind_0, ..., Ind_{k-1})$ with a length of k, where $Ind_{k-1} \in [0, k-1]$ and each Ind_{k-1} is not repeated.[1)]. Then D divides Ind into n parts as P_i's position set $ind_i = (ind_{i,0}, ..., ind_{i,j})$ with a length of j, where $j = k/n$. D then samples M according to ind_i, that is, for the secret share $[\![m]\!]_i$, assign $[\![M]\!][ind_{i,j}]$ to the corresponding $[\![m]\!]_i[ind_{i,j}]$, and the other values in $[\![m]\!]_i$ remain unchanged and are still equal to 0 (Lines 3-9 in Algorithm A1). Finally D distributes $[\![m]\!]_i$ to each participant P_i.

Next, we take two parties as an example to explain the process of generating secret shares. Given a secret $M = [100101]_2$, D will generate two secret shares $[\![m]\!]_0$ and $[\![m]\!]_1$. First D initializes $[\![m]\!]_0 = [000000]$, $[\![m]\!]_1 = [000000]$, then scrambles the sequence $[0, 1, 2, 3, 4, 5]$ and divides it into 2 position sets, $ind_0 = [0, 1, 5]$, $ind_1 = [2, 3, 4]$. Then D samples M, then we have $[\![m]\!]_0[ind_{0,0}] = M[ind_{0,0}] \rightarrow [\![m]\!]_0[0] = M[0] = 1$ and $[\![m]\!]_0[1] = 0$, $[\![m]\!]_0[5] = 1$ can be deduced in the same way. And the value of $[\![m]\!]_0[2]$, $[\![m]\!]_0[3]$ and $[\![m]\!]_0[4]$ are still equal to 0, and finally we have $[\![m]\!]_0 = [100001]$, $[\![m]\!]_1 = [000100]$.

Algorithm A1 Gradient Sharing GS(·)

Input: Secret $M = (m_0, ..., m_{k-1})_2$
Output: Secret sharing $[\![m]\!]_0, ..., [\![m]\!]_{n-1}$

1: The secret distributor D initializes n secret shares $[\![m]\!]_i = (0_0, ..., 0_{k-1})_2$ of M;
2: D randomly reorders the sequence with values from 0 to $k-1$ to get Ind, and then divides Ind into n parts of position set $ind_i = (ind_{i,0}, ..., ind_{i,j})$ for each P_i on average.
3: **for** $i = 0, 1, ..., n-1$ **do**
4: **for** $t = 0, 1, ..., k-1$ **do**
5: **if** $t \in ind_i$ **then**
6: $[\![m]\!]_i[t] \leftarrow [\![M]\!][t]$;
7: **end if**
8: **end for**
9: **end for**
10: **return** $[\![m]\!]_0, ..., [\![m]\!]_{n-1}$.

Algorithm A2 Preprocessing Operation PO(·)

Input: Current global model w^τ
Output: Preprocessed gradient $[\![\nabla w_i^e]\!]_0, [\![\nabla w_i^e]\!]_1$

1: $w_i^\tau \leftarrow \text{LocalUpdate}(w^\tau)$
2: $\nabla w_i^\tau = w^\tau - w_i^\tau$
3: **if** $\nabla w_i^\tau > 0$ **then**
4: $\nabla w_i^q \leftarrow 1$
5: **else**
6: $\nabla w_i^q \leftarrow -1$
7: **end if**
8: $\nabla w_i^e \leftarrow \lceil (1 - \nabla w_i^q)/2 \rceil$
9: $[\![\nabla w_i^e]\!]_0, [\![\nabla w_i^e]\!]_1 \leftarrow F.GS(\nabla w_i^e)$
10: Send $[\![\nabla w_i^e]\!]_0, [\![\nabla w_i^e]\!]_1$ to $\mathcal{S}_0$ and $\mathcal{S}_1$ respectively
11: **return** $[\![\nabla w_i^e]\!]_0, [\![\nabla w_i^e]\!]_1$

* Corresponding author (email: lengtao@iie.ac.cn)

1) Here we use the Fisher-Yates shuffling algorithm to generate a random sequence.

Appendix A.2 GMW

We use GMW protocol [1] to construct Garbled Circuit (GC) [2] for two-party communication. The GMW protocol consists of three stages and its objective function is composed of XOR gates, AND gates and NOT gates. Assume there are two parties P_0, P_1 holding two bit strings $x = (x_0, ..., x_{k-1})_2$, $y = (y_0, ..., y_{k-1})_2$ of length k respectively. The implementation of GMW is described below.

1. *Circuit generation.* The generator needs to convert functions into circuits composed of XOR gates, AND gates and NOT gates.
2. *Secret sharing.* P_0 and P_1 generate secret shares $[\![x]\!] = ([\![x_0]\!], ..., [\![x_{k-1}]\!])$, $[\![y]\!] = ([\![y_0]\!], ..., [\![y_{k-1}]\!])$ of x and y respectively and then send one of the shared shares to each other.
3. *Circuit execution.* P_0 and P_1 input secret shares into the circuit respectively and then execute each gate in the circuit in sequence. We describe their execution processes for different gates, where $[\![z]\!] = ([\![z_0]\!], ..., [\![z_{k-1}]\!])$ is the value to be calculated.
   - P_j executes the XOR gate by calculating $[\![z_i]\!]_j = [\![x_i]\!]_j \oplus [\![y_i]\!]_j$ locally, where $0 \leqslant i \leqslant k-1$, $j = 0, 1$.
   - P_0 executes NOT gate by calculating $[\![z_i]\!]_j = [\![x_i]\!]_j \oplus j$ locally.
   - To execute the AND gate, P_0 first lists the truth table and then P_0 and P_1 perform the calculation together through OT protocol.

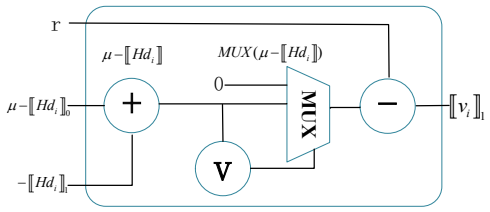


Figure A1 Garbled circuit for calculating client weight. $[\![v_i]\!]_0 = r$, $[\![v_i]\!]_1 = MUX(\mu - [\![Hd_i]\!]) - r$, where r is a random number generated by $\mathcal{S}_0$ and $MUX()$ is a two-to-one data selector. When the input parameter is greater than 0, garbled circuit return $\mu - [\![Hd_i]\!]$, otherwise the output is 0.

Appendix A.3 Generate Multiplication Triples

we use Correlated Oblivious Product Evaluation (COPE) [3] to securely generate $[\![a]\!], [\![b]\!], [\![c]\!]$.

COPE is a protocol based on OT. Its purpose is to allow multiple participants to obtain the secret shares of the product of all input values through COPE after their respective inputs, note that the data held by each participant can only be known by themselves in Algorithm 4. Next, we will take two participants as an example to illustrate the whole process in detail. Suppose there are two parties P_0,P_1 holding a, $b \in F_p$ respectively and they aim to calculate $[\![c]\!] = [\![a]\!] * [\![b]\!]$, where p is a large prime number.

1. P_0 takes a as input and P_1 first converts b to binary stream $(b_0, b_1, ..., b_{k-1}) \in (0,1)^k$, $b = \sum_i^k b_i * 2^{i-1}$ and then takes it as input.
2. P_0 and P_1 execute the OT protocol k times. Note that P_0 is the sender, P_1 is the receiver, r_i is a random value selected by P_0 from F_p in each round of OT protocol (Line 4 in Algorithm A3).
3. P_0 and P_1 compute the secret shares $[\![c]\!]_0, [\![c]\!]_1$ of $[\![c]\!]$ respectively (Lines 6-7 in Algorithm A3).

Algorithm A3 COPE($\cdot$)

Input: P_0 has a, P_1 has b
Output: $[\![c]\!] = a * b$
1: P_1 converts b to binary $(b_0, b_1, ..., b_{k-1})$;
2: **for** $i = 0, 1, ..., k-1$ **do**
3: P_0 generates a random number r_i;
4: P_0 and P_1 jointly execute $q_i \leftarrow OT(r_i, r_i + a, b_i)$;
5: **end for**
6: P_0 computes $[\![c]\!]_0 \leftarrow -\sum_i^k t_i * 2^{i-1}$ as a secret share of $[\![c]\!]$;
7: P_1 computes $[\![c]\!]_1 \leftarrow \sum_i^k q_i * 2^{i-1}$ as a secret share of $[\![c]\!]$;
8: **return** $[\![c]\!]$.

Finally, in the offline phase, we use COPE to generate $[\![a]\!], [\![b]\!], [\![c]\!]$ as shown in Algorithm A4 and the steps are as follows:

1. Each participant P_i randomly generates a random number $[\![a]\!]_i, [\![b]\!]_i \in F_p$, where $[\![a]\!]_i, [\![b]\!]_i$ are the secret shares of $[\![a]\!], [\![b]\!]$ respectively.
2. The COPE protocol is implemented between two different parties. Participants P_i, P_j respectively take $[\![a]\!]_i, [\![b]\!]_j$ as the inputs of COPE, P_i, P_j obtain $[\![t]\!]_{i,j}, [\![q]\!]_{j,i}$ respectively (Line 5 in Algorithm A4).
3. Each P_i calculates the secret share of $[\![c]\!]$ by Eq.A1 (Line 9 in Algorithm A4).

$$[\![c]\!]_i = [\![a]\!]_i * [\![b]\!]_i + \sum_j^n \left([\![t]\!]_{i,j} + [\![q]\!]_{j,i} \right). \tag{A1}$$

Algorithm A4 Generate Multiplication Triples GMT(·)

Input: $\{P_i|0 \leqslant i \leqslant n-1\}$
Output: Multiplication triples $[\![a]\!], [\![b]\!], [\![c]\!]$
1: Each P_i generates a pair of random numbers $\left([\![a]\!]_i, [\![b]\!]_i\right)$ as secret share of $[\![a]\!], [\![b]\!]$;
2: **for** $i = 0, ...n-1$ **do**
3: **for** $j = 0, ...n-1$ **do**
4: **if** $i \neq j$ **then**
5: $[\![t]\!]_{i,j}, [\![q]\!]_{j,i} \leftarrow COPE([\![a]\!]_i, [\![b]\!]_j)$;
6: **end if**
7: **end for**
8: **end for**
9: Each P_i gets $[\![c]\!]_i \leftarrow [\![a]\!]_i * [\![b]\!]_i + \sum_j^n \left([\![t]\!]_{i,j} + [\![q]\!]_{j,i}\right)$;
10: **return** $[\![a]\!], [\![b]\!], [\![c]\!]$.

Appendix A.4 secMul

Secret sharing [4] is a technique for sharing secrets among n participants $\{P_i|i \in [0, n-1]\}$. Specifically, given an additive secret sharing $[\![x]\!] = \sum_{i=0}^{n-1} [\![x]\!]_i$, where $[\![x]\!]_i$ is the share held by participant P_i, an additive secret sharing usually has the following four security operations.

- Given secret shares $[\![x]\!] = \sum_{i=0}^{n-1} [\![x]\!]_i, [\![y]\!] = \sum_{i=0}^{n-1} [\![y]\!]_i$, we have $[\![z]\!] = [\![x]\!] + [\![y]\!] \rightarrow \sum_{i=0}^{n-1} ([\![x]\!]_i + [\![y]\!]_i)$.
- Given two secret shares $[\![x]\!]$, $[\![y]\!]$, we can get $[\![z]\!] = [\![x]\!] * [\![y]\!]$ by multiplying triples [34] as shown in Algorithm A5, where $[\![a]\!], [\![b]\!], [\![c]\!]$ are multiplicative triples.
- Given a secret share $[\![x]\!] = \sum_{i=0}^{n-1} [\![x]\!]_i$ and a constant c, we have $[\![z]\!] = [\![x]\!] + c \rightarrow \sum_{i=0}^{n-1} [\![x]\!]_i + c$.
- Given a secret share $[\![x]\!]$ and a constant c, we have $[\![z]\!] = [\![x]\!] * c \rightarrow \sum_{i=0}^{n-1} ([\![x]\!]_i * c)$.

Algorithm A5 Secure Multiplication SecMul(·)

Input: Secret shares $[\![x]\!]$,$[\![y]\!]$,$[\![a]\!], [\![b]\!], [\![c]\!]$
Output: Secret share $[\![z]\!] = [\![x]\!] * [\![y]\!]$
1: **for** $i \in [0, n-1]$ **do**
2: P_i computes $[\![\varepsilon]\!]_i = [\![x]\!]_i - [\![a]\!]_i$ and $[\![\rho]\!]_i = [\![y]\!]_i - [\![b]\!]_i$;
3: Specify P_0 as the main participant;
4: P_i receives the pair $\left([\![\varepsilon]\!]_j, [\![\rho]\!]_j\right)$ from $P_j, j \neq i$;
5: **if** $i == 0$ **then**
6: $[\![z]\!]_i \leftarrow [\![c]\!]_i + [\![\rho]\!] * [\![a]\!]_i + [\![\varepsilon]\!] * [\![b]\!]_i + [\![\varepsilon]\!] * [\![\rho]\!]$;
7: **else**
8: $[\![z]\!]_i \leftarrow [\![c]\!]_i + [\![\rho]\!] * [\![a]\!]_i + [\![\varepsilon]\!] * [\![b]\!]_i$;
9: **end if**
10: **end for**
11: **return** $[\![z]\!]_i$.

Appendix B Theoretical Analysis

Appendix B.1 Robustness Analysis

Similar to [5], we give a robustness analysis of Hamming distance. Cosine similarity is one of the best metrics for measuring the similarity between two vectors and here we show that using Hamming distance to detect malicious models can achieve equivalent results as cosine similarity by Eq.B1, where cs_i is the cosine similarity of the gradient g_i^q and the root gradient g_s^q.

$$cs_i = \frac{\left\langle \nabla w_i^q, \nabla w_s^q \right\rangle}{\left\| \nabla w_i^q \right\| \cdot \left\| \nabla w_s^q \right\|}. \tag{B1}$$

According to Eq.**??**, we can get Eq.B2, where d is the length of ∇w_i.

$$\begin{aligned} cs_i &= \frac{\sum_{j=1}^{d} \nabla w_{i,j}^q \cdot \nabla w_{s,j}^q}{\sqrt{d} \cdot \sqrt{d}} = \frac{1}{d} \cdot \left(\sum_{j=1}^{d} \left(1 - 2 * \nabla w_{i,j}^e\right) \cdot \left(1 - 2 * \nabla w_{s,j}^e\right) \right) \\ &= 1 - \frac{2}{d} \cdot \left(\sum_{j=1}^{d} \nabla w_{i,j}^e \oplus \nabla w_{s,j}^e \right) = 1 - 2 \cdot \frac{Hd_i}{d}. \end{aligned} \tag{B2}$$

Therefore, we can deduce that when the threshold is $\tau = d/2$, the client weight v_i and cs_i are consistent, implying $v_i > 0 \Leftrightarrow cs_i > 0$.

$$cs_i > 0 \Leftrightarrow 1 - 2 \cdot \frac{Hd_i}{d} > 0 \Leftrightarrow Hd_i < \frac{d}{2} \tag{B3}$$

Based on the above, this scheme uses the Hamming distance to detect malicious models to ensure that the robustness of the scheme is feasible. In addition, using Hamming distance will be more flexible than using cosine similarity, and we can vary the value of the threshold τ for different task metrics to obtain the best Byzantine robustness.

Appendix B.2 Security Analysis

Now given two secret distributors D_0, D_1 hold secrets $M_0 = (M_{0,0}, ..., M_{0,k})_2$, $M_1 = (M_{1,0}, ..., M_{1,k})_2$ respectively, and two participants P_0, P_1. First, D_0 and D_1 jointly negotiate a random seed through secret key exchange technology and digital signature algorithm, and use the shuffling algorithm to randomly sort the sequence $[0, ..., k-1]$ locally, and obtain the same sequence Ind. After that, D_0 and D_1 divide Ind into two parts, namely $ind_0 = (Ind_0, ..., Ind_s)$ and $ind_1 = (Ind_{s+1}, ..., Ind_k)$, where $s = \left\lfloor \frac{k-1}{2} \right\rfloor$. Finally, D_0 samples M_0 according to ind_0 and ind_1 to obtain $[\![M_0]\!]_0 = (M_{0,0}, ..., M_{0,Ind_s}, ..., 0)_2$ and $[\![M_0]\!]_1 = \left(0, ..., M_{0,Ind_{s+1}}, ..., M_{0,k}\right)_2$. Similarly, D_1 generates $[\![M_1]\!]_0 = (M_{1,0}, ..., M_{1,Ind_s}, ..., 0)_2$ and $[\![M_1]\!]_1 = \left(0, ..., M_{1,Ind_{s+1}}, ..., M_{1,k}\right)_2$. Among them, P_0 holds $[\![M_0]\!]_0$, $[\![M_1]\!]_0$, and P_0 holds $[\![M_1]\!]_0$, $[\![M_1]\!]_1$. We analyze the correctness of the sample-based secret sharing method, as Theorem 1 and Theorem 2.

Table C1 Model descriptions of MNIST(Fashion-MNIST) and CIFAR-10.

MNIST	Conv_1	in_channels=1,out_channels=10,kernel_size=5
	Conv_2	in_channels=10,out_channels=20,kernel_size=5
	FC_1	in_features=320, out_features=50
	FC_2	in_features=50, out_features=10
CIFAR-10	Conv_1	in_channels=3,out_channels=6,kernel_size=5,padding=1
	Pool	kernel_size=2, stride=2
	Conv_2	in_channels=6,out_channels=16,kernel_size=5,padding=1
	FC_1	in_features=16*5*5,out_features=120
	FC_2	in_features=120,out_features=84
	FC_3	in_features=84,out_features=10

Theorem 1 (Correctness of additive properties)**.** The sample-based secret sharing methods have all the properties of additive secret sharing, as described in sec. 3.2.

Proof. Given $[\![M_0]\!]_0$ and $[\![M_0]\!]_1$, we have

$$[\![M_0]\!]_0 + [\![M_0]\!]_1 = \left(M_{0,0}, ..., M_{0,Ind_s}, ..., M_{0,Ind_k}\right)_2 = M_0 \,. \tag{B4}$$

According to Eq.B4, given a constant c, we have $\left([\![M_0]\!]_0 + c\right) + [\![M_0]\!]_1 = \left([\![M_0]\!]_0 + [\![M_0]\!]_1\right) + c = M_0 + c$ and $[\![M_0]\!]_0 \cdot c + [\![M_0]\!]_1 \cdot c = \left([\![M_0]\!]_0 + [\![M_0]\!]_1\right) \cdot c = M_0 \cdot c$. From the above, the sampled-secret sharing can be used in Algorithm A5, such as calculating $[\![M_0 \cdot M_1]\!] = [\![M_0]\!] \cdot [\![M_1]\!]$ through $[\![a]\!], [\![b]\!], [\![c]\!]$.

Theorem 2 (Correctness of Hamming distance)**.** The sample-based secret sharing can correctly calculate $Hd = [\![Hd]\!]_0 + [\![Hd]\!]_1$, where the Hamming distance Hd is between M_0 and M_1.

Proof. Then we prove Theorem 2. as follows:

$$\begin{aligned} &[\![Hd]\!]_0 + [\![Hd]\!]_1 = H\left([\![M_0]\!]_0 \oplus [\![M_1]\!]_0\right) + H\left([\![M_0]\!]_1 \oplus [\![M_1]\!]_1\right) \\ &= H\left(M_{0,0} \oplus M_{1,0}, ..., M_{0,Ind_s} \oplus M_{1,Ind_s}, ..., M_{0,Ind_k} \oplus M_{1,Ind_k}\right) = Hd \end{aligned} \,. \tag{B5}$$

The security of sample-based secret sharing mainly depends on the randomness and privacy of Ind. We use secure key exchange technology and digital signatures to negotiate random seeds to ensure that the random seeds will not be leaked, thereby ensuring that Ind cannot be obtained by the adversary. Secondly, we use a secure shuffling algorithm to ensure the randomness of Ind so that Ind can only be cracked violently by adversary with negligible probability.[2)]

In addition, we analyze the privacy of FESD against semi-honest adversaries (servers), In Theorem 3.

Theorem 3 (Privacy of FESD)**.** FESD guarantees that semi-honest adversaries can learn nothing except what they can infer from the aggregated result $([\![\nabla w_i^\tau]\!]_0, [\![\nabla w_i^\tau]\!]_1)$ with probability $1 - \varepsilon$.

Proof. Since the security of any combination of different protocols is guaranteed in the general composability framework [6], we only need to prove the security of a single protocol.

Preprocessing. In this stage, $\mathcal{C}$ and $\mathcal{S}_0$ perform operations locally without interaction. In the real world, $\mathcal{C}$'s view is $\left\{w^t, \nabla w_i^q, [\![\nabla w_i^e]\!], [\![\nabla w_i^e]\!]_0, [\![\nabla w_i^e]\!]_1\right\}$ and $\mathcal{S}_0$'s view is $\{w^t, \nabla w_s^q, [\![\nabla w_s^e]\!], [\![\nabla w_s^e]\!]_0, [\![\nabla w_s^e]\!]_1\}$. In an ideal world, since $\mathcal{C}$ and $\mathcal{S}_0$ can perform preprocessing locally and Sim does not need to simulate this process, their views are consistent with those in the real world, thereby satisfying privacy security. Similarly, in calculating Hamming distance, decoding and model aggregation, each participant performs operations locally without interaction. These stages ensure privacy security by local operations.

Calculate client weight. In this stage, each participant obtains their weights by using garbled circuits which are zero-knowledge and secure. The view composed of tags in the real world by each participant is indistinguishable from that simulated by Sim. This stage ensures privacy security by garbled circuits.

Multiplication. In this stage, $\mathcal{S}_0$ and $\mathcal{S}_1$ generate multiplication triples securely through COPE protocol and local operations. In addition, the COPE protocol has proved to be secure in MASCOT [3], so the multiplication process is secure by using multiplication triples.

Therefore, we guarantee that the adversaries learns nothing beyond what can be inferred from the aggregated results with an overwhelming probability.

Appendix C Performance Analysis

Appendix C.1 Experiment Setup

This solution is implemented in Python 3.8. In the training model stage, we rely on TorchVision and Torch two libraries to implement. In addition, unless otherwise specified, we test the CNN model structure used in the MNIST (Fashion-MNIST) and CIFAR-10 datasets as described in Table C1.

Appendix C.1.1 *Dataset*

We test our scheme with multiple datasets under Byzantine attacks, including 3 datasets for image classification. It should be noted here that all our experiments are under the IID setting. We distribute datasets in the same way that datasets are distributed among clients in FlTrust. Assume that there are M classes in the dataset, we first randomly divide clients into M groups and then set the probability of assigning the dataset with the i-th label to the i-th group as p, note that the probability of assigning it to other groups as $(1-p)/(M-1)$. In the same group, the dataset will be evenly distributed to clients.

The data sets we used in this experiment mainly include MNIST [7], Fashion-MNIST [8] and CIFAR-10 [9]. Next we will briefly introduce root dataset.

Root dataset: For the server, a root dataset is required to obtain a trusted gradient. By default, we assume that the root dataset and the client's local training data have the same distribution, specifically the samples of the root dataset are random and

2) Here we use Diffie–Hellman and an RSA-based signature algorithm to jointly negotiate a secret seed.

uniform from the local clean training data of all clients. For example, we randomly and uniformly sample the root dataset from its 60,000 training samples when using MNIST.

Appendix C.1.2 *Experiment Parameters*

We assume that a total of N clients participate in federated learning and the number of communication rounds is T, where α is the global model learning rate. And the proportion of clients participating in each round of communication is set to σ, which includes malicious clients and non-malicious clients. Among them, corrupt clients account for a proportion of δ of all clients in this round. For local model training, the local model is updated using momentum SGD. In addition, the local batch size is B, the number of local updates is E, the local learning rate is β and the momentum parameter is γ. The test parameter settings we use by default are shown in Table C2.

Table C2 System parameter settings.

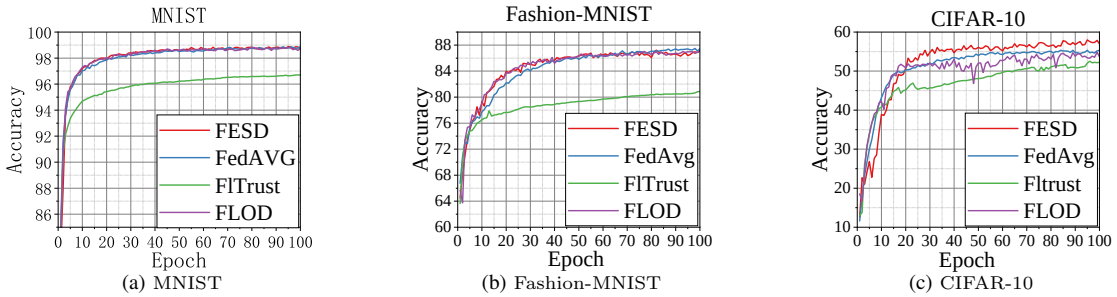
Notation	Parameter	Notation	Parameter
N	100	σ	0.1
E	5	T	100
B	10	α	0.25
β	0.01	δ	0.1

Appendix C.2 Experimental Results

In this comparison experiment, we will use four baseline comparison solutions, namely FedAvg, FLTrust, FLOD and FLDetector. Among them, FedAvg is the basic algorithm of federated learning. The latter three solutions detect malicious clients by calculating cosine similarity, Hamming distance, and Euclidean distance respectively, and support dishonest-majority settings. To test the effect of this scheme on these three design goals (Section 4.4), we carry out the following experiments.

Appendix C.2.1 *Fidelity*

A Byzantine robust aggregation method should also be suitable for attack-free situations in practical applications. When $\delta = 0$, it should have almost no loss in model performance. Therefore, we first conduct model training on the FedAvg scheme without malicious model attacks. When FedAvg achieves the best performance, we compare FESD with it to show how good or bad this scheme is in terms of fidelity. In addition, we also measure the accuracies of FlTrust and FLOD. In the test, we successively use three different datasets, as shown in Figure C1(a) and Figure C1(b), when we use the MNIST and Fashion-MNIST datasets for training, the accuracy of this scheme is almost the same as the FedAvg and FLOD schemes and higher than FlTrust. And as shown in Figure C1(c), the accuracy of FESD is even higher than those of other schemes. This is because the root model update in FESD also participates in model aggregation, which is more conducive to model training to a certain extent and the effect is more prominent on the CIFAR-10 dataset, which is closer to the real dataset and difficult to identify. Finally, we present the highest accuracy and loss measured by each scheme in Table C3. Based on the above, the effect of this scheme in terms of fidelity is feasible.



(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

Figure C1 Model performance testing without attacks

Appendix C.2.2 *Robustness*

In order to fully evaluate the Byzantine robust performance of this scheme, we continuously change the value of δ of corrupt clients from 0.1 to 0.8 and compare the model accuracy measured in different datasets when compared with the FlTrust and FLDetector. It should be noted that the value of the global learning rate α is also different in the case of different proportions of corrupt clients.

We show the model accuracy for different proportions of corrupt clients in Figure C2. First of all, we find that FESD shows good stability under the MNIST, Fashion-MNIST and Cifar-10 datasets. When $\delta = 0.1$, the model accuracy on the MNIST dataset is 98.75%, when $\delta = 0.8$, the accuracy of the global model remains above 98.37% and only decreases by 0.38%. Similarly, in the Fashion-MNITS dataset, the model accuracy does not vary by more than 2%. In the Cifar-10 dataset, the model accuracy does not vary by more than 5%. In addition, under different proportions of corrupt clients, the model accuracy of FESD is generally equal to or higher than that of FlTrust, FLOD and FLDetector, which shows that it is effective for us to participate in the global model aggregation process of root model update, which can improve the model accuracy to a certain extent. In addition, when $\delta > 0.5$,

Table C3 Accuracy and loss without malicious attack.

$\delta = 0$	dataset	FESD	FedAvg	FlTrust	FLOD
Accuracy	MNIST	98.889	98.879	96.71	98.779
	Fashion-MNIST	87.220	87.489	80.86	87.040
	CIFAR-10	57.990	55.290	52.951	55.819
Loss	MNIST	0.033	0.037	0.103	0.038
	Fashion-MNIST	0.376	0.345	0.507	0.385
	CIFAR-10	1.793	1.828	1.942	1.839

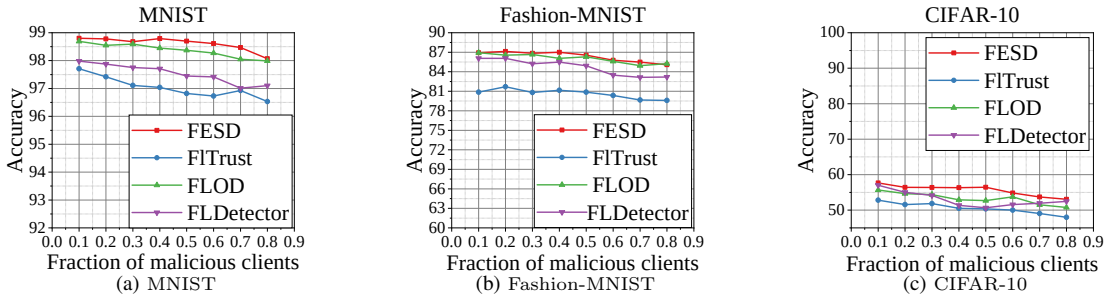


Figure C2 Model performance test with different fraction of corrupt clients

Table C4 Client overhead with communication rounds

Dataset		FESD			FdeAvg		
		MNIST	Fashion	CIFAR-10	MNIST	Fashion	CIFAR-10
Rounds	10	169.942s	166.817s	146.022s	163.118s	165.895s	144.157s
	50	849.716s	835.687s	731.673s	816.732s	829.783s	721.931s
	100	1699.672s	1672.458s	1463.796s	1635.986s	1661.342s	1445.426s

the trend of model accuracy decline is slower than other schemes. This shows that it is effective for us to use the proportion of detected corrupt clients to adaptively scale the global learning rate and to some extent make up for the problem of missing effective information caused by malicious clients. In addition, when a malicious client is detected by FLDetector, the malicious client will be directly removed from the model aggregation. Although it effectively prevents the influence of malicious data on the model, it also causes the data information of some clients to be missing, which leads to the convergence speed of FLDetector being much lower than that of FESD, FlTrust and FLOD. In Figure C2, FESD, FlTrust and FLOD basically converge in 100 rounds, while FLDetecor needs more than 300 rounds to complete convergence. Based on the above, the robustness of the scheme is feasible.

Table C5 The overhead of several schemes on PCBit2A

Operation		Bit2A					
Model		FNet			ResNET-18		
Phase	Rounds	FESD	FLOD	FLGUARD	FESD	FLOD	FLGUARD
Online	10	–	1.43s	2.90s	–	7.44s	15.28s
	50	–	7.44	14.77s	–	33.86s	91.74s
	100	–	14.07s	30.60s	–	67.93s	153.13s
Offline	10	–	58.16s	151.22s	–	237.17s	616.64s
	50	–	290.06s	754.16s	–	1185.06s	3081.16s
	100	–	580.16s	1508.42s	–	2370.45s	6163.17s

Appendix C.2.3 *Efficiency*

Since the client in federated learning is a resource-constrained device, it is necessary to minimize its load. To measure user efficiency, we use client runtime as a metric. We give the total time for all clients to run locally under different rounds in Table C4, from which we can calculate the average time of each client. After calculation, it can be concluded that the average duration of each client in this solution or FedAvg does not exceed 0.03s. On the premise of ensuring data privacy and security, it is desirable not to increase the user load too much. In addition to highlighting the advantages of this scheme, we also measure the extra overhead used by the FLOD scheme at PCBit2A. It can be seen from Table C5 that this scheme adopts the sampling secret sharing method, which can greatly reduce the computational cost and improve efficiency.

References

1 O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proc. ACM Symposium on the Theory of Computing (STOC'87)*, 1987, pp. 218–229.
2 A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. IEEE Annual Symposium on Foundations of Computer Science (FOCS'86)*, 1986, pp. 162–167.
3 M. Keller, E. Orsini, and P. Scholl, "Mascot: faster malicious arithmetic secure computation with oblivious transfer," in *Proc. ACM Conference on Computer and Communications Security (CCS'16)*, 2016, pp. 830–842.
4 J. Cha, S. K. Singh, T. W. Kim, and J. H. Park, "Blockchain-empowered cloud architecture based on secret sharing for smart city," *Journal of Information Security and Applications*, 2021, doi:https://doi.org/10.1016/j.jisa.2020.102686.
5 Y. Dong, X. Chen, K. Li, D. Wang, and S. Zeng, "Flod: Oblivious defender for private byzantine-robust federated learning with dishonest-majority," in *European Symposium on Research in Computer Security (ESORICS'21)*, 2021, pp. 497–518.
6 R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Pro. IEEE Annual Symposium on Foundations of Computer Science (FOCS'01)*, 2001, pp. 136–145.
7 L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.
8 H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint*, 2017, arXiv:1708.07747.
9 A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.