ISSN: 2582-8118 ***Volume 3, Issue 1; Jan 2023***

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYSIS IN SCIENCE AND ENGINEERING

Web: https://www.iarj.in/index.php/ijrase/index

1. A Multi-Trapdoor Editable Blockchain Scheme Based on Shamir Secret Sharing for Federated Learning

XueWei, ChungenXu, LeiXu, LinMei, PanZhang

School of Mathematics and Statistics,
Nanjing University of Science and Technology,
Nanjing, Jiangsu, China.

ABSTRACT

Federated learning is a popular learning mechanism. A server coordinates many clients to complete the training process of the model. However, there is a problem of poisoning attacks in federated learning. For example, the client may send malicious parameters to damage the global model's performance. To reduce the clients' dishonest behavior during training, some schemes are committed to scoring the clients' parameters to get the reputation value and storing the reputation value on the blockchain to achieve the clients' incentive and further reduce the poisoning attack during the federated learning process.

However, due to the non-editable nature of the blockchain, when the reputation value stored on the blockchain is incorrect, it cannot be modified. This problem leads to the unfairness of the scheme that relies on traditional blockchain to store reputation value to motivate clients. We improved the original chameleon hash function to solve this problem and proposed a new multi-trap editable blockchain scheme.

Besides, we combined the dynamic Shamir secret sharing technology to ensure the trap door's security and avoid trap door centralization. We use our editable blockchain to store the reputation value generated in federated learning. It not only realizes the client's incentive but also effectively reduces dishonest behavior during training and supports the modification of the reputation value on the blockchain.

Our scheme ensures the client can participate fairly in the federated learning training task. Experiments show that our editable blockchain scheme supports the safe modification of reputation value and has significant function advantages compared with existing schemes.

KEYWORDS

Federated Learning, Blockchain, Shamir Secret Sharing, Chameleon Hash Function.

Introduction:

In the era of big data, Internet companies and individual users create a large amount of data daily. These data are of great value. Using these data for training can obtain corresponding models, which can be used for prediction, classification, etc12. However, in real life, more than the data of a single company or user is often needed to support the training of a model with good performance. Obtaining a good performance model often requires a large amount of data to participate in training.

Therefore, to get a better model, many enterprises will collect a large amount of data for unified training, which is machine learning3456, That is, the data of multiple companies or users are collected together and trained by a centralized central cloud server. However, collecting a large amount of data in real life is often a considerable challenge.

On the one hand, in most industries, data exist in the form of islands. Due to industrial competition, national policies and other issues, even among different departments of the same company, there are still many obstacles to achieving data integration. In reality, it is almost impossible to integrate data scattered in different places and institutions, Or the cost is enormous. On the other hand, in machine learning, users or enterprises need to send their data to the central server, and the central server will use these data for unified training to get the final model. However, personal data contains a lot of private information, and the direct upload of data will cause many privacy disclosure problems. Therefore, many users or companies prefer to keep their sensitive data private from a third party.

For example, companies or research institutions often face the difficulty of collecting enough personal medical data in healthcare applications because patients want their privacy to be kept private. Lack of training data will lead to poor performance of the model finally trained. To avoid directly sharing sensitive data, Federated Learning789 emerged at the historic moment.

Federated learning, also known as collaborative learning, distributed learning. In federated learning, all participants (either companies or individuals) collaborate to train a shared model under the coordination of a trusted central server without disclosing their sensitive data information. In this process, each participant only needs to use their data to train the model locally and upload the intermediate parameters to the central cloud server. The central cloud server aggregates the received parameters and sends the aggregation results to the participants to continue the model's training process. Obviously, in the process of federal learning, the participants' sensitive data is still kept locally, protecting data privacy.

However, in federated learning, the client that used to be responsible for providing data is now a participant involved in the whole training process. We can't guarantee that each client will faithfully perform the local training process, nor can we guarantee that malicious adversaries will not control the client to produce dishonest behavior. Previous research has proved that the adversary can disrupt the whole training process through poisoning attacks10111213, this will affect the final model's performance. Such operations may also indirectly infringe on clients' data. For example, by uploading the inverted and amplified gradient, the opponent can infer whether the sample is used for training the target model14. Therefore, this poses new challenges to federal learning.

At present, some defense mechanisms have been proposed against poisoning attacks. For example, the Multi-krum scheme15 mainly identifies parameters which significantly differ from the parameters and treats them as malicious parameters. Precisely, the Euclidean distance between the client and other parameters is calculated and summed in each iteration. The client's parameters with the minimum Euclidean distance will as the aggregation parameters and returned to the client participating in the training. Contra16 detects the parameters of malicious clients by evaluating the angle difference of model parameters between clients; GeoMed17 uses the difference of Lp norm between client parameters to identify malicious parameters. Khazbak et al.18 used cosine similarity between gradients to identify malicious clients. Clients considered malicious would not participate in the model's aggregation process. Liu et al.19 used the Pearson coefficient between the client model parameters and the average value of all client model parameters to identify malicious clients and assigned a lower weight to the identified malicious clients in the aggregation process to reduce their impact.

However, these schemes often limit the number of malicious clients. When the number of malicious clients is too large, these methods will fail. In order to make up for the shortcomings of the above schemes, some scholars combine federated learning with blockchain, use reputation as an indicator to evaluate the reliability or credibility of the client according to its past behavior202122, and store the reputation value on the blockchain, so that the subsequent task initiator can select the client by referring to the reputation value on the blockchain before the model training task starts,

For example, Zhang et al.23 used model cross entropy to evaluate the model quality of clients and used it as a measure of reputation evaluation, and stored the reputation value on the public chain for subsequent reference; Kang et al.24 also designed a reputation based client selection scheme. Unlike the previous scheme, this scheme mainly uses the alliance blockchain to manage the reputation value of clients, and selects clients based on the reputation value. However, although the existing schemes can effectively calculate the reputation value of the client and store the reputation value on the blockchain, so as to ensure that before the start of the federated learning task, the task initiator can judge the reliability of the client by referring to the corresponding reputation value of the client on the blockchain and then select the client, the above schemes store the reputation value on the traditional blockchain, Due to the non-editable nature of blockchain, when a malicious adversary masquerades as an honest client and produces bad behavior, the client's reputation value will be low and unmodifiable, which will lead to a lower probability that the client will participate in other model training tasks than other clients.

To solve the above challenges, we introduce editable blockchain technology into joint learning for the first time in this paper. We have implemented an editable blockchain federated learning scheme, which uses blockchain to store reputation values and support secure modification of reputation values. Specifically, we have made the following three contributions:

- Based on the chameleon hash function, we propose a new editable blockchain scheme to ensure the modifiability of reputation value on the blockchain. We combine the editable blockchain with federated learning for the first time to realize the incentive for clients.

- We combine Shamir secret sharing technology to ensure the security of the trapdoor in the reputation value modification process and avoid the trapdoor centralisation problem in reputation value modification.
- We simulated the modification scheme of reputation value on our editable blockchain and compared it with the existing scheme. The experiment shows that our scheme's performance is in an acceptable range and has significant advantages in function compared with the current scheme.

Next, we will introduce our scheme from the definition of the scheme model, design objectives, scheme construction, security analysis and other aspects.

2. Background Knowledge:

Next, we will briefly introduce the background knowledge required for our work.

2.1 Federated Learning:

Federated learning aims to train a classifier F (such as image classification) through distributed training so that F can correctly classify the new sample. Federated learning was initially proposed by Google to solve the problem of android terminal users updating their models locally. In federated learning, each client's data will not leave the local and only need to upload the model parameters obtained by local training.

Federated learning is usually participated by one server S and N clients. Server S is mainly responsible for aggregating the model parameters of the client, and each client C_i has a local private data set $D_i = \{x_1^i, x_2^i, ..., x_{l_i}^i\}$ with a size of l_i, l_i is the total amount of data involved in the training.

In each round of training, the client will use the global model w of the previous round and its local data for training. Specifically, the client starts from w_G^{t-1}, running the stochastic gradient descent(SGD) algorithm on each data group to minimize the corresponding loss function and get the local model parameter u_i^t, then calculate the model update $w_i^t = u_i^t - w_G^{t-1}$ and sends it to the server. Then the server aggregates $w_G^t = w_G^{t-1} + \sum_{i=1}^{N} \alpha_i w_i^t$, where $\alpha_i = \frac{l_i}{l}$. The training process will be repeated until reaching the pre-specified training rounds or the model accuracy reaches the pre-required standard.

2.2 Additive Secret Sharing:

The primary purpose of additive secret sharing25 is to share a secret s between two parties. We choose a random value r, share r with the first party, and share $s - r$ with the second party. Obviously, the two secret shares are random so no one can calculate relevant information about the secret s. As long as the two parties add their respective secret shares, the secret s will be revealed.

The additive secret sharing scheme consists of the following algorithms:

Sharing Algorithm(Shr(·)). Assuming that the client U has a secret value of u with l bits, then U selects a random value $r \in Z_{2^l}$, and then calculates the two secret shares of u, respectively: $\langle u\rangle_0 = (u-r)\ mod\ 2^l$, $\langle u\rangle_1 = r$. Then send $\langle u\rangle_0$, $\langle u\rangle_1$ to two servers, CS_0 and CS_1.

Reconfiguration Algorithm(Rec(·)). If a third party S needs to recover the secret value u, CS_0 sends its own secret share $\langle u\rangle_0$ to S, and CS_1 also sends its secret share $\langle u\rangle_1$ to S, then S can calculate $u = \langle u\rangle_0 + \langle u\rangle_1$ to get the secret value.

Here is an introduction to the addition and multiplication of secret values using the addition secret sharing scheme.

Addition. Suppose the two secret values are u and v, among them, the secret shares of secret value u are $\langle u\rangle_0$ and $\langle u\rangle_1$, the secret shares of the secret value v are $\langle v\rangle_0$ and $\langle v\rangle_1$, server CS_0 owns $\langle u\rangle_0$ and $\langle v\rangle_0$, CS_1 owns $\langle u\rangle_1$ and $\langle v\rangle_1$. In order to calculate $u+v$, CS_0 calculate $\langle u+v\rangle_0 = \langle u\rangle_0 + \langle v\rangle_0$, CS_1 calculate $\langle u+v\rangle_1 = \langle u\rangle_1 + \langle v\rangle_1$, CS_0 and CS_1 exchange the secret share of $u+v$, then calculate $\langle u+v\rangle_0 + \langle u+v\rangle_1 = \langle u\rangle_0 + \langle v\rangle_0 + \langle u\rangle_1 + \langle v\rangle_1 = u+v$. It is also true when there are N secret values. Therefore, the addition secret sharing satisfies the addition homomorphism.

Multiplication. Suppose that the two secret values are u and v, where the secret shares of the secret value u are $\langle u\rangle_0$ and $\langle u\rangle_1$. The secret shares of the secret value v are $\langle v\rangle_0$ and $\langle v\rangle_1$. In order to calculate uv, we need a pre-defined multiplicative triplet (a,b,c), where a,b is randomly selected in Z_{2^l}, and $c = ab\ mod\ 2^l$. Server CS_i has the secret shares corresponding to multiplication triples, $\langle a\rangle_i$, $\langle b\rangle_i$, $\langle c\rangle_i$, and then each server calculates $\langle \alpha\rangle_i = \langle u\rangle_i - \langle a\rangle_i$, $\langle \beta\rangle_i = \langle v\rangle_i - \langle b\rangle_i$. The two servers collaborative calculate $\alpha = \langle \alpha\rangle_0 + \langle \alpha\rangle_1$, $\beta = \langle \beta\rangle_0 + \langle \beta\rangle_1$, then CS_0 calculate $\langle \gamma\rangle_0 = \beta\langle \alpha\rangle_0 + \alpha\langle \beta\rangle_0 + \langle c\rangle_0$, CS_1 calculate $\langle \gamma\rangle_1 = \alpha\beta + \beta\langle \alpha\rangle_1 + \alpha\langle \beta\rangle_1 + \langle c\rangle_1$, then CS_0 and CS_1 collaborative calculate $\gamma = \langle \gamma\rangle_0 + \langle \gamma\rangle_1 = uv$. When $u = v$, we can get u^2 through the above calculation method.

2.3 Chameleon Hash Function based on Discrete Logarithm:

Assume that q, N are two large prime numbers and that $N = kq+1$, Z_N^* is a group of order q, and g is the generator of the group Z_N^*. We set trap door $sk \in Z_N^*$, public key $pk = g^{sk}\ mod\ N$.

$Hash(pk,m,r,g)$: Enter the public key pk and give a message $m \in Z_N^*$ and a random value $r \in Z_N^*$, we can get chameleon hash $Hash(pk,m,r,g) = g^m pk^r \ mod \ N$.

$Forge(sk,m,r,m')$: Enter trapdoor sk, original message m, random number r and new message m', and output a new random number r', which can satisfy $ch = Hash(pk,m',r',g) = g^{m'} pk^{r'} \ mod \ N$.

Because of $Hash(pk,m,r,g) = Hash(pk,m',r',g)$, so we can ge $g^m pk^r = g^{m'} pk^{r'}$, then, we can get $m + skr = m + skr'$, then $r' = \dfrac{m-m'}{sk} + r \ mod \ N$.

2.4 Editable Blockchain:

Blockchain26 is a chain-like structure that stores data in blocks and chronologically

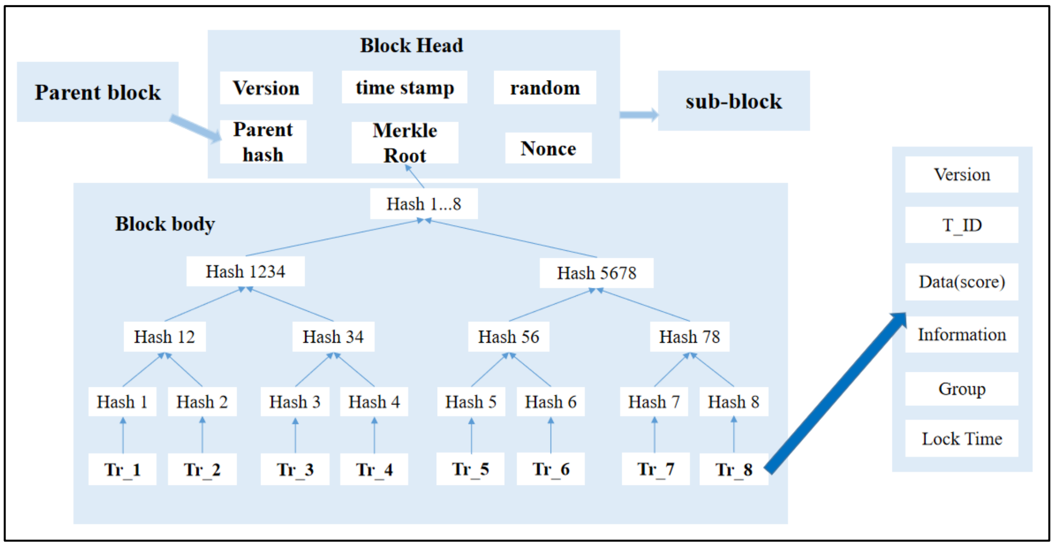


Figure 1 : Block Composition

connects blocks from end to end. It collectively maintains a reliable distributed database in a decentralized and untrusted way and simultaneously maintains a distributed ledger using technologies such as timestamp and hash function.

As shown in Figure 1, each block contains the block header and block body, where the block header contains the parent block hash, version number, timestamp, difficulty value, random number and Merkle root. The block body contains the information of all transactions in this block. The corresponding hash value of these transactions constitutes a Merkle tree, which can be used to verify the authenticity of data quickly. The value of the Merkle root is stored in the block header. In the blockchain, the information contained in the transaction is shown in Figure 1: "Version" refers to the version number of the transaction, "T-ID" refers to the unique identifier of the transaction in the entire blockchain network, "Data" is used to storing transaction information, "Information" refers to the relevant information used in the process of calculating the transaction, "Group" refers to the miner collection for verifying the transaction, and "Lock Time" refers to the transaction lock time.

The editable blockchain scheme27 mainly uses the physical editing method and is mainly used in data storage. In the editable blockchain, the block header still uses the traditional hash calculation. In contrast, the hash value of the transaction in the Merkle tree of the block body is calculated by the chameleon hash function, which makes it convenient for us to achieve transaction-level editing accuracy. In the editable blockchain, the trapdoor owner can calculate the conflict of the chameleon hash in polynomial time to ensure that the hash value before and after transaction editing remains unchanged and then ensure that the Merkle tree root information in the block header remains unchanged.

3. Problem Statement:

3.1 System Model:

In our system, there are five basic entities:

- Key Distribution Center(KGC): KGC is a fully trusted entity. KGC is mainly responsible for generating two pairs of public and private keys $(sk_1, pk_1),(sk_2, pk_2)$ required for the chameleon hash function and broadcasting the public keys pk_1 and pk_2. Then KGC generates the corresponding secret polynomial of the threshold t according to Shamir secret sharing scheme and randomly selects x_i, calculating the $i-th$ secret share $(x_i, Shr_1(x_i), Shr_2(x_i), i=1,2,\cdots,n)$ of sk_1 and sk_2 through Shamir secret sharing scheme and distributing $(Shr_1(x_i), Shr_2(x_i))$ to supervisor R_i. KGC will go offline after distributing the key and secret share.
- Cloud Server(CS): Our system includes a cloud server, which is recorded as CS. The server will truthfully collect parameter updates from the client, then complete the calculation process of reputation value and return the aggregated parameters.
- Remote Client(C_i): The client is the owner of the training data in federated learning. They train the model on their local private data and then send their parameters to the cloud server. The server CS will verify the parameters and get the corresponding reputation value. However, some clients may be controlled by enemies or misbehave due to other external reasons. In addition, we assume that the data held by all clients are independent and distributed, which is the same as many previous works2829.
- Miner(M_i): Miners are mainly responsible for chaining the reputation value generated by the cloud server.
- Set of Regulators(R, R_i): Our system includes a set of supervisors, including a leader (R) and n supervisor (R_i). The supervisors cooperate with the leader to complete the modification of the reputation value stored on the blockchain. We also assume that supervisors and the leader are honest but curious, which means they will perform their

tasks honestly but still be curious about the data they obtained. In addition, we assume that leaders will not collude with supervisors, and at most $t-1$ supervisors will collude.

3.2 Threat Model and Security Objectives:

Threat Model: In this chapter, we mainly consider external adversaries. External malicious adversaries will attempt to tamper with the reputation value stored on the blockchain according to the characteristics of the editable blockchain. For example, a client will try to change the reputation value on the blockchain to a higher reputation value to have a greater chance to participate in the model training of other task initiators. Some adversaries also try to change the lower reputation value on the blockchain to a higher reputation value or attempt to interfere with the model's training process and tamper with the reputation value of other clients.

Security Objectives: Our security goal is to ensure the safe modification of the reputation value. Any external adversary has no privilege to modify the reputation value. The leader and supervisor can only modify the reputation value in cooperation, and the leader and supervisor cannot complete the modification of the reputation value alone unless they cooperate. To achieve this goal, in our scheme, we use Shamir secret sharing technology to divide the trap door into n shares and send them to n supervisors to ensure that neither supervisors nor leaders will know the specific value of the trap door.

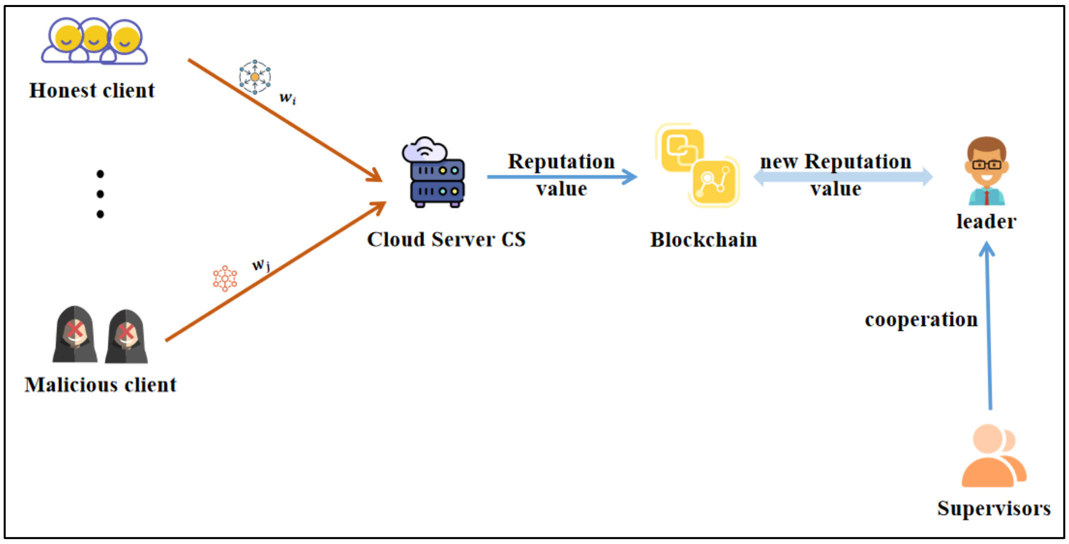


Figure 2 : System Architecture

4. Scheme Construction:

In this section, we first discuss the overview of our proposed scheme. The overall structure of our scheme is shown in Figure 2. In addition, the symbols and related descriptions used in this chapter are listed in Table 1 for reference.

Table 1 : Parameter Distribution

sk_1 , sk_2	Two trap doors of editable blockchain
pk_1 , pk_2	Public key corresponding to trap door
m	Reputation value of the client
L	Number of clients
m_{new}	The new reputation value of the client
n	Number of supervisors
t	Threshold value of Shamir secret sharing scheme
$s_i(k)$	The sum of malicious scores obtained from the previous k rounds during training of the client i
$Shr_1(x_i)$	The $i-th$ Shamir secret share of private key sk_1
$Shr_2(x_i)$	The $i-th$ Shamir secret share of private key sk_2

4.1 System Initialization:

The trusted key generation center (KGC) generates two pairs of public and private keys (sk_1, pk_1) and (sk_2, pk_2) required for the editable blockchain. Then KGC broadcasts the public key pk_1 and pk_2, selects the secret polynomial according to the threshold value t, and calculates the commitment value corresponding to the polynomial and broadcast it to ensure that the supervisor can verify the secret share after receiving it. Next, KGC uses the Shamir secret sharing scheme to divide sk_1 and sk_2 into n shares and distribute them to n supervisors. KGC will go offline after distributing the key and secret share. At the same time, all miners received the public key pk_1 and pk_2. In addition, at the beginning of training, CS initializes the global model w_{init} randomly.

Specifically, let q and N be two large prime numbers and satisfy $N = kq+1$, Z_N^* is the group of order q, and g is the generator of the group Z_N^*, (N, g) is public. First, KGC generated the public and private key pairs (sk_1, pk_1) and (sk_2, pk_2), among, $pk_1 = g^{sk_1}$ and $pk_2 = g^{sk_2}$, sk_1 and sk_2 are two trap doors. Then KGC selects the secret polynomial $Shr_1(x) = sk_1 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{t-1}x^{t-1}$, $Shr_2(x) = sk_2 + b_1x + b_2x^2 + b_3x^3 + \cdots + b_{t-1}x^{t-1}$, calculates the commitment value $C_1 = (g^{sk_1}, g^{a_1}, \cdots, g^{a_{t-1}})$, $C_2 = (g^{sk_2}, g^{b_1}, \cdots, g^{b_{t-1}})$ in advance, and public it. Then KGC calculates the secret shares of trapdoor sk_1 and sk_2, which are $\{s_1(x_i) = (x_i, Shr_1(x_i))\}_{i=1}^{n}$, $\{s_2(x_i) = (x_i, Shr_2(x_i))\}_{i=1}^{n}$, and sends $(x_i, s_1(x_i), s_2(x_i))$ to the supervisor R_i and $(x_1, x_2, \cdots, x_n)$ to the leader R.

After receiving the secret share value $(x_i, s_1(x_i), s_2(x_i))$ and commitment C_1, C_2, supervisor R_i verifies whether the equations of $g^{Shr_1(x_i)} = \prod_{j=0}^{t-1}\left(g^{a_j}\right)^{x_i^j}$, $g^{Shr_2(x_i)} = \prod_{j=0}^{t-1}\left(g^{b_j}\right)^{x_i^j}$ are true. If it is true, it means that the received secret share is indeed distributed by KGC and has not been tampered with by the adversary. Then the supervisor will save their secret share. When the reputation value needs to be modified, supervisors will use their secret share to modify the reputation value.

In addition, each system node has a unique ID, and there is a specific random number function library to provide the parameter for the chameleon hash function.

4.2 Calculate Reputation Value:

Client local training: The client's local training process includes the local training and the parameter sending stage. Specifically, client C_i uses the dataset to train the local model and get the local parameter w_i and sends it to CS.

Calculate malicious score: Since the calculation of the reputation value is not the focus of this paper, we use the cosine similarity between client parameters to calculate the reputation value. Precisely, after receiving local parameters, CS will calculate $s = \frac{1}{L}\sum_{i=1, i\neq q}^{L} \frac{w_p \cdot w_i}{\| w_p \| \| w_i \|}$ to get the malicious score.

Since training tasks usually require multiple rounds, storing malicious scores of each round of the client will cause high storage costs and great inconvenience to the reference of the task initiator. So we take the average of all malicious scores obtained during training as the final client's reputation value. Calculating reputation value is shown in the Algorithm 1.

Specifically, the server CS initializes the client's reputation value as 0 and then obtains the reputation value vector $(s_1(0), s_2(0), \cdots, s_L(0)) = (0, 0, \cdots, 0)$ composed of the reputation values of all clients. Then before the parameter aggregation of the k round training, CS calculates the sum of the cosine similarity of the client parameters to get the client's malicious score s. Then CS updates the client CS's reputation value $s_i(k) = s_i(k-1) + s$ and obtains the reputation value vector $(s_1(k), s_2(k), \cdots, s_L(k))$ of the k round. When the training is over, CS calculates $s_i = \frac{s_i(K)}{K}$ to get the reputation value of client C_i.

Algorithm 1 : Reputation value calculation algorithm

Input: Client reputation value vector $(s_1(0), s_2(0), \cdots, s_L(0))$, Reputation value of the clients s, Training round K, Number of clients L.

Output: $(s_1, s_2, \cdots, s_L)$.

```
1.    $CS_0$ initialize the reputation value vector of the client
      $(s_1(0), s_2(0), \cdots, s_L(0)) = (0, 0, \cdots, 0)$;
2.    FOR $K = 1$ to $K$
3.      FOR $i = 1$ to $L$
4.        $CS_0$ calculates the malicious score $s$ of $C_i$;
5.        $CS_0$ updates $C_i$'s reputation value: $s_i(k) = s_i(k-1) + s$;
6.    END FOR
7.      END FOR
8.    FOR $i = 1$ to $L$
9.      $CS_0$ calculates: $s_i = \frac{s_i(K)}{K}$
10.   END FOR
11.   RETURN $(s_1, s_2, \cdots, s_L)$
```

4.3 Reputation Value Storage Up Chain:

Let us first introduce the composition of each transaction in the blockchain, as shown in Figure 1. "Version" refers to the version number of the transaction, "T_ID" refers to the unique identifier of the transaction in the entire blockchain network, "Data" is used to store the transaction information, which mainly refers to the reputation value of the client, "Information" refers to the relevant information used in the calculation of the transaction, "Group" refers to the ID of the set of Shamir secret shares randomly selected by the system, that is, the ID set of the supervisor, and "Lock Time" refers to the transaction lock time.

Specifically, the process of the reputation value chain is divided into the following steps:

Step 1: After the server generates the reputation value m, the miner will calculate g^m and use the random number r_1 generated by the system to calculate $H(m, r_1, pk_1, pk_2) = g^m \cdot pk_1^{r_1} \cdot pk_2 = g^{m+sk_1 \cdot r_1 + sk_2}$, where the reputation value m is stored in "Data" and r_1, pk_1, pk_2 is stored in "Information" and generate transactions.

Step 2: After the system generates the transaction, the transaction is sent to each miner node through the P2P network to form a unified transaction pool to prepare for reaching consensus. Other miner nodes will verify whether the received reputation value is legal. After the verification, a block will be generated to record the verified reputation value, and then proof of work(PoW)2630 will be run to reach a consensus. Finally, the generated blocks will be added to the blockchain, also known as distributed ledgers, which means that the reputation value is successfully stored on the blockchain.

4.4 Reputation Value Modifications:

Algorithm 2 : Reputation value modification algorithm

Input: new reputation value m_{new}, random number a

Output: r_2

```
1.  $KGC$ generates the private key $sk$ and calculate the corresponding secret share $\{s(x_i)=(x_i, Shr(x_i))\}_{i=1}^{n}$ according to the secret polynomial $Shr(x)$;
2.  $KGC$ sends $s(x_i)$ to $R_i$;
3.  Leader $R$ randomly selects $t$ values from $(x_1, x_2, \cdots, x_n)$, and calculates:
4.      $b_i = \prod_{j=1, j\neq i}^{t} \frac{(-1)^t x_j}{x_i - x_j}$;
5.  Leader $R$ calculates the additive secret share $\langle m-m_{new}\rangle_i$ of $m-m_{new}$ ;
6.  Leader $R$ sends $b_i$ and $\langle m-m_{new}\rangle_i$ to the corresponding supervisor $R_i$;
7.  $R_i$ calculate:
8.      $s_1(i) = shr_1(i)\cdot b_i \cdot a$
9.      $s_2(i) = shr_2(i)\cdot b_i \cdot a$;
10.     $s(i) = shr(i)\cdot b_i \cdot a$;
11.     $m(i) = \langle m-m_{new}\rangle_i \cdot a$;
12.     $secret(i) = m(i) + s_2(i) - s(i)$;
13. $R_i$ sends $s_1(i)$ and $secret(i)$ to the leader $R$;
14. Leader $R$ calculates:
15.     $sum_1 = \sum_{i=1}^{t} s_1(i)$;
16.     $sum_2 = \sum_{i=1}^{t} \sec ret(i)$;
17.     $r_2 = \frac{sum_2}{sum_1} + r_1$;
18. RETURN $r_2$
```

After the reputation value is linked, the trap door owner can modify the content on the blockchain due to the characteristics of the editable blockchain. However, there is an excellent risk of centralization when the trap door belongs to only one person. In addition, the editable blockchain scheme based on the discrete logarithmic chameleon hash function has a deadly problem. Because the information stored on the blockchain is open and transparent, when the message m_1 and m_{new}, random number r_1 and r_2 is known, the trap door will be easy to infer, which also leads to other people also can modify the information. To avoid these two problems, we designed a multi-trap door scheme combined with Shamir secret sharing scheme. On the one hand, it ensured that the trap door would not belong to one person, avoiding the single-point problem, and on the other hand, we designed a multi-trap door scheme to ensure the security of the trap door. In addition, to prevent malicious adversaries from launching man-in-the-middle attacks, we assume that the messages sent between leaders and supervisors are sent after the sign, which will not be explained below. The process of reputation value modification is described in detail as follows:

Step 1: The leader initiates the transaction editing request. The request includes information such as editing reason, editing content, etc. Then the leader broadcasts the editing request to the whole network. The supervisor participating in the transaction will verify the validity of the editing request. If the authentication passes, continue to step 2. Otherwise, the application will be rejected.

Step 2: After the application is approved, the supervisor will request the KGC. KGC generates a new public and private key (pk, sk) and distributes the secret share corresponding to sk to the supervisor.

Step 3: After receiving the secret share of the new private key sk, the supervisor cooperates with the leader to calculate the edited parameter r and keep the chameleon hash value of the transaction unchanged. First, according to the characteristics of chameleon hash, We can get $H(m, r_1, pk_1, pk_2) = g^m \cdot pk_1^{r_1} \cdot pk_2 = g^{m+sk_1 \cdot r_1 + sk_2}$, $H(m_{new}, r_2, pk_1, pk) = g^{m_{new}} \cdot pk_1^{r_2} \cdot pk = g^{m_{new}+sk_1 \cdot r_2 + sk}$, So the new random number r_2 can be calculated as follows: $r_2 = \frac{m_1 - m_{new} + sk_2 - sk}{sk_1} + r_1$.

The calculation process refers to the Algorithm 2, which is described as follows:

When the reputation value needs to be modified, KGC first generates a new public-private key pair (sk, pk), distributes the secret share of sk to the supervisor, and distributes pk to the leader. Then the leader randomly selects t values from the $(x_1, x_2, \cdots, x_n)$. Here we assume that the leader chooses $x_1, x_2, \cdots, x_t$, then calculates $b_i = \prod_{j=1, j\neq i}^{t} \frac{(-1)^t x_j}{x_i - x_j}$ and the additive secret share $\langle m - m_{new} \rangle_i (i = 1, 2, \cdots, t)$ of $m - m_{new}$, and sends b_i and $\langle m - m_{new} \rangle_i$ to the corresponding supervisor R_i. After receiving b_i and $\langle m - m_{new} \rangle_i$, each supervisor R_i calculates $shr_1(i) \cdot b_i$, $shr_2(i) \cdot b_i$ and $shr(i) \cdot b_i$, according to their secret share. To avoid the leader recovering the trap door value based on the secret value, all supervisors will use the same random number generator to generate the same random number a, then calculate $s_1(i) = shr_1(i) \cdot b_i \cdot a$, $s_2(i) = shr_2(i) \cdot b_i \cdot a$, $s(i) = shr(i) \cdot b_i \cdot a$, and calculate $m(i) = \langle m - m_{new} \rangle_i \cdot a$, then calculate $s_2(i)$, $s(i)$ and $m(i)$ to get $secret(i) = m(i) + s_2(i) - s(i)$, and send $s_1(i)$

and $secret(i)$ to the leader R. Because the leader owns $secret(i) = m(i) + s_2(i) - s(i) = (\langle m - m_{new} \rangle_i + shr_2(i) \cdot b_i - shr(i) \cdot b_i) \cdot a$, they cannot get the information about the trap door, but leaders can calculate $sum_1 = \sum_{i=1}^{t} s_1(i)$, $sum_2 = \sum_{i=1}^{t} secret(i)$, and leaders can calculate $r_2 = \frac{sum_2}{sum_1} + r_1$, there is $Hash(m, r_1, pk_1, pk_2) = Hash(m_{new}, r_2, pk_1, pk)$, so using r_2, leaders can successfully modify the reputation value m, and during the process, leaders cannot obtain any information about the double-trap door.

The secret shares will be updated regularly to prevent malicious adversaries from attacking the secret shares held by the supervisor. Specifically, KGC updates a secret polynomial $h(x) = h_1 x + h_2 x^2 + h_3 x^3 + \cdots + h_{t-1} x^{t-1}$, calculates the commitment value $C = \left(g^{a_1}, \cdots, g^{a_{t-1}}\right)$, and makes it public. Then KGC calculates the corresponding secret shares $\{hs_i = \left(x_i, h\left(x_i\right)\right)\}_{i=1}^{n}$ and sends them to the supervisor R_i. the supervisor R_i will verify the commitment after receiving the new secret share and update their secret share value $\left(x_i, h\left(x_i\right) + Shr_1\left(x_i\right), h\left(x_i\right) + Shr_2\left(x_i\right)\right)$ after the verification, thus completing the security update of the secret share.

Step 4: Broadcast r_2 and m_{new} to the entire network, and all miner nodes update the transaction data on the chain.

According to the characteristics of the chameleon hash, when we use the editable blockchain to store the reputation value generated during training, for other external adversaries, because the trap door is not known, the chameleon hash still has the anti-collision characteristics, which also ensures the reliability of the reputation value on the blockchain.

Before the start of other training tasks, the task initiator can refer to the reputation value stored on the blockchain to select the client and select the client with a high reputation value to participate in the training, which realizes the incentive to the client.

When the reputation value stored on the blockchain is incorrect, the leader and the supervisor can cooperate to modify the reputation value due to the characteristics of the chameleon hash function. At the same time, our scheme can reduce the impact of malicious adversaries and ensure that benign clients can participate in the model training fairly.

5. Security Proof:

5.1 Correctness Proof:

First, according to the characteristics of chameleon hash, we have $H(m, r_1, pk_1, pk_2) = H(m_{new}, r_2, pk_1, pk)$, which means $g^{m+sk_1 \cdot r_1 + sk_2} = g^{m_{new}+sk_1 \cdot r_1 + sk}$. Then we can get: $m + sk_1 \cdot r_1 + sk_2 = m_{new} + sk_1 \cdot r_1 + sk$. Therefore, the new random number $r_2 = \frac{m_1 - m_{new} + sk_2 - sk}{sk_1}$. So we only need to prove that the r_2 calculated in the Algorithm 2 is the above form.

In the Algorithm 2, sum_1 is calculated as follows:

$$sum_1 = \sum_{i=1}^{t} s_1(i) = \sum_{i=1}^{t} shr_1(i) \cdot \prod_{j=1, j\neq i}^{t} \frac{(-1)^t x_j}{x_i - x_j} \cdot a = sk_1 \cdot a$$

sum_2 is calculated as follows:

$$sum_2 = \sum_{i=1}^{t} secret(i) = \sum_{i=1}^{t} (m(i) + s_2(i) - s(i)) = ((m - m_{new}) + sk_2 - sk) \cdot a$$

Therefore, $\frac{sum_2}{sum_1} + r_1 = \frac{((m - m_{new}) + sk_2 - sk) \cdot a}{sk_1 \cdot a} + r_1 = \frac{(m - m_{new}) + sk_2 - sk}{sk_1} + r_1 = r_2$.

Therefore, calculating the random number r_2 is correct.

5.2 Security Proof:

EB-FL ensures that leaders, supervisors and external adversaries cannot obtain the trap value while modifying the reputation value.

Proof :

In the Algorithm 2, because each supervisor only has the secret share of the trap door, we assume that at most only $t-1$ supervisors may collude. According to the security of the Shamir secret sharing scheme, the supervisor will not obtain any information about the trap door value while modifying the reputation value. Then we prove that leaders will not get any information about trap door value.

During modifying the reputation value, the leader gains $s_1(i)$ and $secret(i)$, namely $s_1(i) = shr_1(i) \cdot b_i \cdot a = < sk_1 >_i \cdot a$ and $secret(i) = m(i) + s_2(i) - s(i) = < (m - m_{new}) + sk_2 - sk >_i \cdot a$.

Obviously, the leader cannot get any information about the trap door from $< sk_1 >_i \cdot a$ and $< (m - m_{new}) + sk_2 - sk >_i \cdot a$. In addition, when calculating r_2, the leader can get $sk_1 \cdot a$ and $((m - m_{new}) + sk_2 - sk) \cdot a$. Because of the random number a, the leader does not know the specific value of the trap door in the whole process. This ensures that each modification of the reputation value needs to be completed by the leader and the supervisor. In the process of modifying the reputation value, the leader and the supervisor will not get the information about the trap door. Therefore, the process of modifying the reputation value is safe. External adversaries can only try to infer the trap value by intercepting the information sent by the leader and supervisors, but the analysis is the same as above. The adversary can only obtain the trap value with a disturbance at most and cannot get the actual value of the trap door, so our process of modifying the reputation value is safe.

6. Experimental Result

This section carries out simulation experiments for our scheme, mainly uses python to construct a simple blockchain, simulates the process of reputation value up-chain and reputation value modification in EB-FL, and calculates its corresponding time cost. Since our experiment is simulated on a single machine, we omit the communication time cost between the supervisor and the leader due to information transmission. We mainly focus on the time cost of calculating the new random number corresponding to the chameleon hash function through secret sharing technology and the time the miners spend to complete the consensus agreement.

6.1 Analysis of Experimental Results:

Experimental setup: In our experiment, we set up ten clients and then defined the number of supervisors as n =6. The parameters involved in the experiment are described as follows: the sizeable prime number q used in the chameleon hash is randomly generated by the system, and its length is guaranteed to be 20. In addition, we need at least three supervisors to participate in modifying the reputation value. That is, the threshold of the Shamir secret sharing scheme is 3. In addition, in the POW consensus protocol, we set the difficult target to satisfy four zeros at the end of the hash value corresponding to the random number.

Evaluation Index:

- Single Score Time: To express our experimental results more clearly, we modify each client's reputation value and obtain the time cost corresponding to the new random number generated for each client to show the corresponding time difference when modifying the reputation value of different clients.
- Multiple Score Time: When modifying reputation values, we sometimes must modify multiple reputation values simultaneously. Here, we use Multiple Score Time to represent the time cost of the leader and supervisors modifying multiple reputation values simultaneously.
- Single Time: We use Single Time to represent the time cost of modifying the reputation value and completing the consensus. That is, the sum of the time cost of the new random number calculated by the leader and the supervisor and the time cost of the miner

completing the POW consensus agreement.

- Multiple Time: We use Multiple Time to represent the time cost of multiple reputation value modifications and completing the consensus. That is, the sum of the time cost of the new random number calculated by the leader and supervisor and the time cost of the miner completing the POW consensus agreement in the multiple reputation value modification process.

6.2 Performance Analysis:

In our experiment, we mainly focus on the time cost of calculating the new random number in the process of transaction modification. After calculating the client's credit score, the client's index, reputation value, and time will be packaged into transactions, which will be stored on the blockchain. When the reputation value is incorrect, the leader and supervisor can work together to modify the reputation value. In Figure 3, we show the time consumed to calculate the new random number corresponding to each client when modifying the reputation value of the client. We can see that the time cost consumed is stable between 0.991 ms ~ 0.998 ms, and the average time is 0.997 ms. We can see that the time cost consumed in calculating the new random number is low, which also shows that our scheme has high efficiency in calculating the random number. In addition, we show the time cost when modifying multiple transactions simultaneously in Figure 4. It can be seen that when only one transaction is modified, the corresponding time cost is about 0.998 ms. In contrast, when two transactions are modified, the corresponding time cost is about 1.975 ms, and when three transactions are modified, the corresponding time cost is about 2.792 ms. It can be seen that with the increase of transactions, the corresponding time cost of modifying transactions is also gradually increasing. This also corresponds to the results in Figure 3.

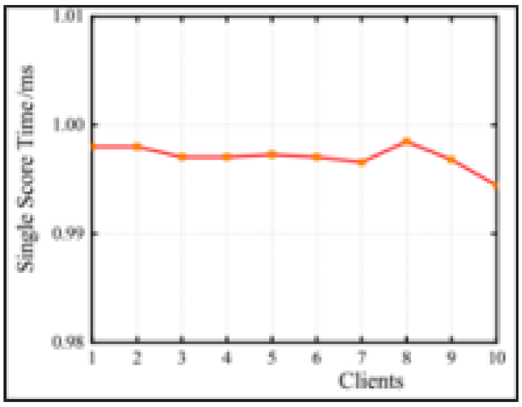


Figure 4 : Blockchain single transaction

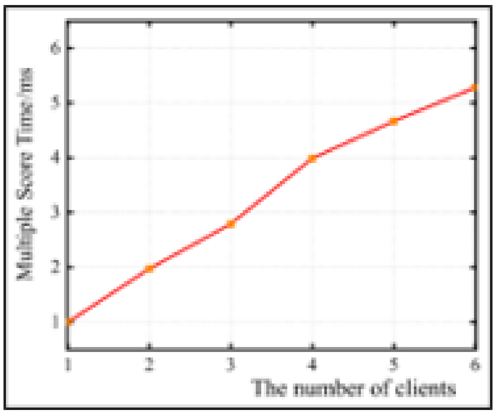
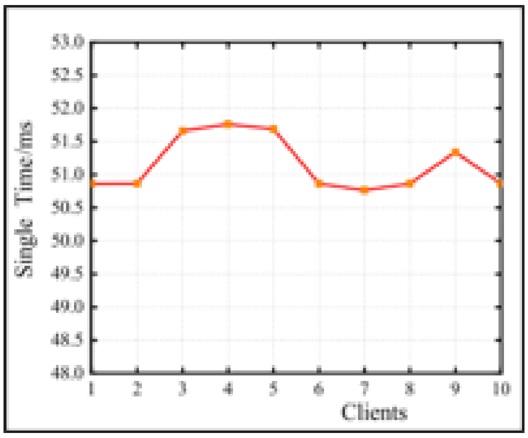


Figure 3 : Blockchain multiple transactions

In addition, miners must reach a consensus after calculating the new random number to modify the new reputation value. Here we use the POW consensus mechanis, and the difficult problem we set is that the last four digits of the hash value corresponding to the random number are 0. Figure 5shows the time cost of calculating the random number and completing the POW consensus in modifying each client's reputation value. It can be seen that although the time cost consumed by the client has an inevitable fluctuation, the overall difference does not exceed 1 ms, and the average time cost is 51.154 ms. Obviously, compared with the time cost of completing the POW consensus, the time cost of calculating the random number can be ignored, which also shows that our reputation value modification scheme will not bring a significant time cost and is generally stable within an acceptable range. Figure 6 shows the time cost of calculating the random number and completing the POW consensus when modifying the reputation value of multiple clients simultaneously. Since miners only need to reach a consensus once when modifying multiple reputation values simultaneously, the increased time cost is only the time cost corresponding to calculating the random number as the number of transactions increases. It can be seen from the figure that when modifie multiple reputation values, the time cost of modifying multiple reputation values at one time will be much lower than the time cost of modifying different reputation values multiple times, because miners only need to reach a consensus once, which greatly reduces the time cost.

The above experiment results show that the time cost caused by EB-FL's reputation value modification is generally stable within an acceptable range.



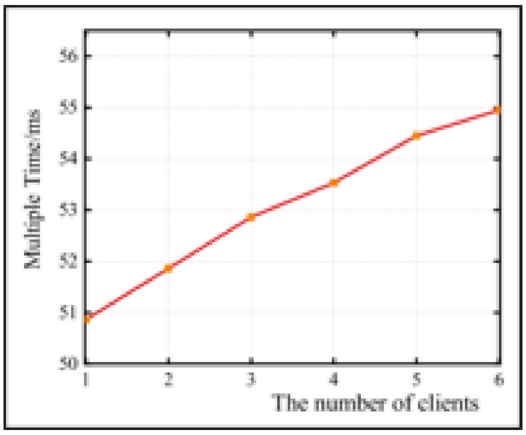


Figure 6 : Blockchain single transactio **Figure 5 : Blockchain multiple transactions**

6.3 Functional Analysis:

In this section, we mainly compare EBSC27, IMRF24, TrustedTe30 to highlight the effect of our scheme, as shown in Table 2.

Table 2 : Scheme comparison

	Reputation value can be modified	**Reputation value can be modified multiple times**	**Safety of trapdoor**
EB-FL	√	√	√
EBSC	√	×	×
IMRFL	×	×	\
TrustedTed	×	×	\

The EBSC scheme mainly uses Shamir secret sharing technology and the original chameleon hash to construct a decentralized editable blockchain scheme. Although it can support the modification of information on the blockchain, the information on the blockchain can only be modified once because the modified trap door value will leak due to the characteristics of the chameleon hash. So it cannot guarantee the security of the trap door, Nor can it guarantee the safe modification of information on the blockchain. IMRFL and TrustedTed both store the client's reputation value in the blockchain, and the difference is that IMRFL adopts the alliance chain, while TrustedTed adopts the public chain. However, the reputation value of these two schemes cannot be modified. In the long run, it cannot achieve true fairness and will inevitably cause high storage costs. In contrast, our scheme EB-FL not only supports the modification of the reputation value but also ensures that the trap door will not leak during the modification of the reputation value, thus ensuring the security of the reputation value.

It can be seen that our scheme has certain advantages in terms of efficiency and function.

7. Summary:

This paper proposes an editable blockchain scheme based on chameleon hash, which mainly improves the chameleon hash function of the single trapdoor. This scheme combined with the Shamir secret sharing technology to design a decentralized editable blockchain with multiple trapdoors. This blockchain is used to store the reputation value during the training process of federated learning, which can support the safe modification of the reputation value. At the same time, for the security and stability of the system, In combination with the dynamic secret sharing scheme, we regularly update the secret component of the trap door owned by the supervisor to ensure that the adversary cannot obtain the trap value by attacking the secret component of the supervisor. We use the editable blockchain to store the reputation value generated during the training process of the federated learning model, which can provide reference for other model training tasks in the future, and can also achieve incentives for the client, which can mitigate dishonest behaviors of clients during model training.

8. References:

1. T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, “Pcanet: A simple deep learning baseline for image classification?” *IEEE transactions on image processing,* vol. 24, no. 12, pp. 5017–5032, 2015.
2. G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal processing magazine,* vol. 29, no. 6, pp. 82–97, 2012.
3. P. Watcharapichat, V. L. Morales, R. C. Fernandez, and P. Pietzuch, “Ako: Decentralised deep learning with partial gradient exchange,” in *The Seventh ACM Symposium on Cloud Computing*. USA: ACM, 2016, pp. 84–97.
4. S. Teerapittayanon, B. McDanel, and H.-T. Kung, “Distributed deep neural networks over the cloud, the edge and end devices,” in *2017 IEEE 37th international conference on distributed computing systems*. USA: IEEE Computer Society, 2017, pp. 328–339.
5. K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and

O. Mutlu, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *14th USENIX Symposium on Networked Systems Design and Implementation*. USA: USENIX Association, 2017, pp. 629–647.

6. J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, "Federated learning for healthcare informatics," *Journal of Healthcare Informatics Research,* vol. 5, no. 1, pp. 1–19, 2021.
7. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-effficient learning of deep networks from decentralized26 data," in *20th International Conference on Artificial Intelligence and Statistics*. USA: PMLR, 2017, pp. 1273–1282.
8. V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," *Advances in neural information processing systems*, vol. 30, pp. 4424–4434, 2017.
9. K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan *et al.*, "Towards federated learning at scale: System design," *Machine Learning and Systems,* vol. 1, pp. 374–388, 2019.
10. C. Fung, C. J. Yoon, and I. Beschastnikh, "The limitations of federated learning in sybil settings," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses*. Spain: USENIX Association, 2020, pp. 301–316.
11. J. Zhang, J. Chen, D. Wu, B. Chen, and S. Yu, "Poisoning attack in federated learning using generative adversarial nets," in *13th IEEE International Conference On Big Data Science And Engineering*. New Zealand: IEEE, 2019, pp. 374–380.
12. E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*. Online: PMLR, 2020, pp. 2938–2948.
13. M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *29th USENIX Security Symposium*. USA: USENIX Association, 2020, pp. 1605–1622.
14. M. Nasr, R. Shokri, and A. Houmansadr, "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*. USA: IEEE, 2019, pp. 739–753.
15. P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," 27 *Advances in Neural Information Processing Systems*, vol. 30, pp. 119– 129, 2017.
16. S. Awan, B. Luo, and F. Li, "Contra: Defending against poisoning attacks in federated learning," in *European Symposium on Research in Computer Security*. Germany: Springer, 2021, pp. 455–475.
17. D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*. Sweden: PMLR, 2018, pp. 5650– 5659.
18. Y. Khazbak, T. Tan, and G. Cao, "Mlguard: Mitigating poisoning attacks in privacy preserving distributed collaborative learning," in *2020 29th International Conference on Computer Communications and Networks*. USA: IEEE, 2020, pp. 1–9.
19. X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, "Privacy-enhanced federated learning against poisoning adversaries," *IEEE Transactions on Information Forensics and Security,* vol. 16, pp. 4574–4588, 2021.
20. J. Kang, Z. Xiong, D. Niyato, D. Ye, D. I. Kim, and J. Zhao, "Toward secure blockchain-enabled internet of vehicles: Optimizing consensus management using

reputation and contract theory," *IEEE Transactions on Vehicular Technology,* vol. 68, no. 3, pp. 2906–2920, 2019.

21. Y. Liu, K. Li, Y. Jin, Y. Zhang, and W. Qu, "A novel reputation computation model based on subjective logic for mobile ad hoc networks," *Future Generation Computer Systems,* vol. 27, no. 5, pp. 547–554, 2011.
22. X. Huang, R. Yu, J. Kang, Z. Xia, and Y. Zhang, "Software defined networking for energy harvesting internet of things," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1389–1399, 2018.
23. Q. Zhang, Q. Ding, J. Zhu, and D. Li, "Blockchain empowered reliable federated learning by worker selection: A trustworthy reputation evaluation method," in *2021 IEEE Wireless Communications and Networking Conference Workshops*. China: IEEE, 2021, pp. 1–6.
24. J. Kang, Z. Xiong, D. Niyato, S. Xie, and J. Zhang, "Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 700–10 714, 2019.
25. I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual Cryptology Conference*. USA: Springer, 2012, pp. 643–662.
26. M. Risius and K. Spohrer, "A blockchain research framework," *Business & Information Systems Engineering*, vol. 59, no. 6, pp. 385–409, 2017.
27. S. Fan and Y. Chen, "Editable blockchain scheme based on shamir chameleon hash secret sharing," in *6th Information Technology and Mechatronics Engineering Conference,* vol. 6. China: IEEE, 2022, pp. 1125–1128.
28. G. Xu, H. Li, Y. Zhang, S. Xu, J. Ning, and R. Deng, "Privacy preserving federated deep learning with irregular users," *IEEE Transactions on Dependable and Secure Computing,* vol. 19, no. 2, pp. 1364– 1381, 2022.
29. G. Baruch, M. Baruch, and Y. Goldberg, "A little is enough: Circumventing defenses for distributed learning," *Advances in Neural Information Processing Systems,* vol. 32, pp. 8632–8642, 2019.
30. M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *Secure information networks*. Belgium: Springer, 1999, pp. 258–272.
31. M. H. Rehman, A. M. Dirir, K. Salah, E. Damiani, and D. Svetinovic, "Trustfed: a framework for fair and trustworthy cross-device federated learning in iiot," *IEEE Transactions on Industrial Informatics,* vol. 17, no. 12, pp. 8485–8494, 2021.