Privacy-Enhanced and Verification-Traceable Aggregation for Federated Learning

Yanli Ren, Yerong Li, Guorui Feng, and Xinpeng Zhang

Abstract—**Federated learning (FL) is a distributed machine learning framework, which allows multiple users to collaboratively train and obtain a global model with high accuracy. Currently, FL is paid more attention by researchers and a growing number of protocols are proposed. This article first analyzes the security vulnerabilities of the VerifyNet and VeriFL protocols, and proposes a new aggregation protocol for FL. We use additive homomorphic encryption and double masking to simultaneously protect the user's local model and the aggregated global model while most of the existing protocols only consider the privacy of the local model. Also, linear homomorphic hash and digital signature are used to achieve traceable verification, which means the users can not only verify the aggregation results, but also be able to identify the wrong epoch if the results are wrong. In summary, our protocol can realize the privacy of the local model and global model and achieve verification traceability even if the cloud server colludes with malicious users. The experimental results show that the proposed protocol improves the security of FL without decreasing the efficiency of the users and classification accuracy of the training model.**

Index Terms—**Federated learning (FL), homomorphic encryption (HE), Internet of Things (IoT) security, privacy-preserving, traceable verification.**

I. Introduction

MACHINE learning (ML) is widely available for Internet of Things (IoT) to improve system performance through calculation on massive data recently. In traditional ML, a large amount of data is stored on a central server, from which ML models can be generated. This is obviously not appropriate for IoT, because it is dangerous to send a large quantity of data created by all mobile devices to the central server for training. In order to prevent privacy disclosure during transmission of data in IoT, federated learning (FL) is proposed by [1]. Its design goal is to protect the privacy of terminal and personal data in the exchange of data. Hence, prediction and classification of targets may become more safer and efficient by combining FL and IoT, especially in the applications of smart transportation, smart healthcare, and smart home.

In FL [2], training data is no longer concentrated on the server, but distributed to other terminal devices of users. Multiple users only need to upload local model parameters to achieve efficient training of the local model without uploading local data sets. Users who have only a small amount of local training data can also improve the accuracy of their models through FL. FL is expected to form the basis of common algorithms and networks for the next generation of artificial intelligence because of the decentralization of training data. However, despite the advantages of FL, such a joint training process is not secure enough. For example, it is shown in [3] that attackers can obtain data of user or other useful information from gradients if user uploads gradients directly to the server. In recent years, many privacy-preserving FL protocols were proposed to strengthen privacy of the users. Most protocols were based on secure multiparty computing (SMC), homomorphic encryption (HE), or differential privacy (DP) techniques.

In general, FL consists of one cloud server (CS) and multiple users, and so it is a multiparty calculation problem [4], [5]. Bonawitz *et al.* [6] proposed a security aggregation protocol applied in ML, which protects the local gradient of users based on SMC and solves the problem of dropping off by using secret sharing. Nevertheless, the communication cost is quite large owing to the multiple interactions among users and the server. Later, Bell *et al.* [7] preprocessed the encryption gradient and reduced the communication cost, but they did not take into account the verifiability of the aggregation results. Xu *et al.* [8] introduced a verifiable aggregation protocol, but the calculation overhead of verification is too large. Recently, Guo *et al.* [9] solved the problem of expensive computation during the process of verification. However, the above protocols cannot protect the privacy of the aggregation results which are public for the server.

In the protocols based on HE, users encrypt the plaintext and send it to the server. The server completes calculation on the ciphertext and returns the result. Then, users decrypt the result to get the corresponding plaintext of the calculation results [10], [11]. For example, Phong *et al.* [12] proposed the additive HE to protect privacy of the local gradient. In this method, the server aggregates the ciphertext of gradient sent by users and returns it to the users. Finally, users decrypt the aggregation result to get the sum of gradients. Since all

Manuscript received 26 June 2022; accepted 25 July 2022. Date of publication 1 August 2022; date of current version 7 December 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 62072295; in part by the Natural Science Foundation of Shanghai under Grant 20ZR1419700 and Grant 22ZR1481000; and in part by the Henan Key Laboratory of Network Cryptography Technology under Grant LNCT2021-A13. *(Corresponding author: Yanli Ren.)*

Yanli Ren is with the School of Communication and Information Engineering, Shanghai University, Shanghai 200444, China, and also with the Henan Key Laboratory of Network Cryptography Technology, Zhengzhou 450040, China (e-mail: renyanli@shu.edu.cn).

Yerong Li, Guorui Feng, and Xinpeng Zhang are with the School of Communication and Information Engineering, Shanghai University, Shanghai 200444, China (e-mail: liyerong@shu.edu.cn; grfeng@shu.edu.cn; xzhang@shu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2022.3194930

2327-4662 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

users use the same key, the server can get their local gradient once it colludes with a malicious user. Boneh *et al.* [13] mitigated this problem by using secret sharing to share decryption keys among all users. However, this protocol complicates the decryption process and greatly increases the computational overhead. Hao *et al.* [14] proposed a method of combing fully HE and DP to realize postquantum security of the gradient. However, it only applies to gradients with fewer dimensions.

On the other hand, DP [15] protects the privacy of the users through random noise [16]. Specifically, Hu *et al.* [17] added random noise on data during the process of updating the gradient, but the accuracy of the training model will be reduced. To strike a balance between accuracy and security, Wei *et al.* [18] proposed a protocol that randomly adjusted the number of users so that the model would be the most accurate at the same security level. Then Chamikara *et al.* [19] came up with a method to perturb the data set rather than the local gradient, while keeping the accuracy. But none of these protocols allow for verifiability of the aggregation results.

Although the above protocol can protect the privacy of the partial model privacy, there are still some security and privacy issues. The first reason is that the trained global gradients appear entirely in plaintext for the server in the previous protocols, which is unacceptable in many cases. Because a malicious attacker or server can get useful information from the global model [20]. The second reason is that once the attacker controls the server and tampers with the global model, the integrity of the global model cannot be guaranteed. That will cause the user to get the wrong model and eventually lead to the wrong classification. The last one is that malicious server is likely to collude with some dishonest users in an effort to access the local model parameters of honest users. Considering the above problems, we propose a new protocol to simultaneously realize the privacy of the local model and global model, and achieve the traceable verifiability of the aggregation results returned by the server.

In detail, we propose a protocol for FL, which can not only verify the aggregation results but also protect the privacy of the users and trace the wrong results. We first adopt an approach based on learning with error (LWE) encryption and double masking to support privacy protection for each user. Then, a secret sharing protocol is used to solve the disconnection problem of the users during training. Finally, the user can verify the aggregation result based on the linear homomorphic hash (LHH) function. A summary of our important contribution is listed as follows.

1) *Privacy:* Our protocol can simultaneously protect the user's local gradients and the aggregated global gradient, even if the attacker colludes with multiple dishonest users. The reason is that the server only aggregates ciphertexts, from which the server and other attackers cannot obtain useful information. We use the LWE and double-masking method for encryption to protect the privacy of gradient during the training process, including the local gradient and global gradient.
2) *Verifiability:* For preventing malicious attackers from tampering with the final aggregation results, we use LHH to verify them. And, we use digital signature to ensure the integrity of LHH. Furthermore, users verify the aggregation results at the end of all iterative training, not in each epoch of training, which can greatly reduce the computational overheads of users during the verification process.
3) *Traceability:* In FL, another important challenge is traceability, which means that the users can further identify the wrong epoch when the aggregation results cannot pass the verification successfully. In the proposed protocol, users first verify the validity of the digital signature before verifying the LHH function. Once the signature is invalid, the hash value of the corresponding user in the current epoch is considered to have been maliciously forged.

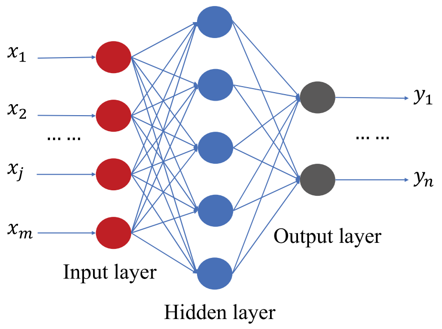

Fig. 1. Neural network.

The remainder of this article is organized as follows. In Section II, we introduce the definitions of deep learning (DL) and FL and some cryptographic primitives. In Section III, we analyze security vulnerabilities of the verification phase in VerifyNet [8] and VeriFL [9] protocols in detail. In Section IV, a privacy-enhanced and verification-traceable aggregation protocols are proposed. Then, we analyze security and efficiency of the proposed protocol in Section V. Next, the experimental results are shown in Section VI. Finally, we conclude this article in Section VII.

II. Definitions and Cryptographic Primitives

In this section, we introduce the definition of DL, the system model, the threat model, and some cryptographic primitives used in our protocol.

A. Brief Introduction of DL

DL is a technique for implementing ML and uses deep neural networks to express features during the learning process. There is a neural network with five hidden layers, m inputs, and n outputs as shown in Fig. 1. In general, the training process of a neural network is completed by iterative feedforward and back propagation. The mathematical operation of a neural network is expressed as a function: $f(x, w) = y$, where x is the data to be predicted, y is the output of neural network's prediction, and w is the internal parameter of the function f.

Suppose the user has a sample of observations as (x_i, y_i) in the training data set $D = \{(x_i, y_i), i = 1, 2, \dots, J\}$. And, the

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

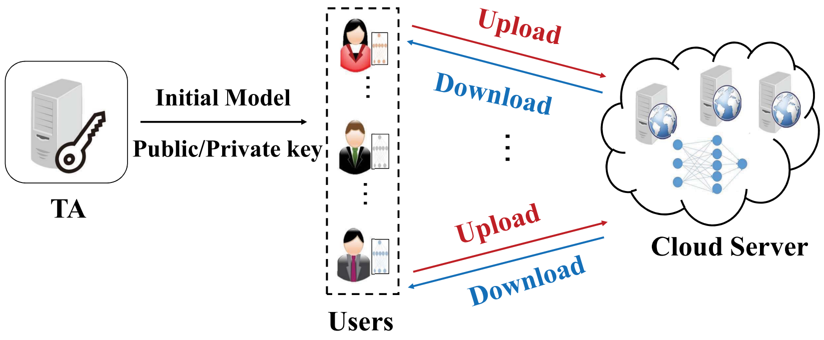

Fig. 2. System model.

loss function in the learning model can be defined as follows:

$$L(D, w) = \frac{1}{|D|} \sum_{(x_i, y_i) \in D} l(y_i, f(x_i, w)). \tag{1}$$

The difference between the predicted and actual values of the model can be evaluated by the loss function in DL. Smaller loss functions indicate better robustness of the model. In back propagation, the stochastic gradient descent (SGD) [21] is utilized to find the minimum of the loss function by adjusting the parameter w

$$w^{k+1} \leftarrow w^k - Ir\nabla_w L_f\left(D^k, w^k\right). \tag{2}$$

Ir is the learning rate, $D^k \subseteq D$ is a subset of the training data set, and w^k is the parameter after the kth iteration.

In practice, a large amount of training data will be stored in different mobile devices rather than centralized in one user. So FL was proposed, which improves the efficiency and quality of training and protects the privacy of local training sets. In FL, each user $P_i \in N$ has a private local data set $D_{P_i} \in D$. Therefore, SGD in FL is rewritten

$$w^{k+1} \leftarrow w^k - Ir\frac{\sum_{P_i \in N} \nabla_w L_f\left(D^k_{P_i}, w^k\right)}{|N|}. \tag{3}$$

B. System Model

FL is a very effective ML framework based on DL and proposed by Google [1]. By combining training data sets and learning models on multiple mobile devices, it can train an effective and accurate model. The privacy of these distributed data will not be disclosed, because the users upload the parameters of their own model rather than the original data during training, such as gradients. However, the attacker can obtain useful information related to the data set of users through the gradients if the gradients are transmitted in a public channel. In order to protect the privacy of the users, the gradients need to be encrypted before transmission. Therefore, compared to the general FL model, our system model consists of not only a CS and multiple users, but also a trusted authority (TA) as shown in Fig. 2. The purpose of TA is to generate public and private key pairs and initialize public parameters. It will be offline when it has completed these jobs. Such a system model including a TA is also widespread in most of privacy-preserving FL, e.g., [8], [14], and [28]. The model training process includes the following steps.

1) *Initialization:* TA initializes the entire neural network model, generates public parameters, and distributes public and private key pairs to each user.
2) *Training Local Model:* Each user updates the local model according to the initial parameters and their training data set, and then sends the ciphertext of the local model parameters to CS.
3) *Updating Global Model:* CS aggregates the ciphertext of the user's local model parameters to get the ciphertext of the global model. Finally, the users download the ciphertext, decrypt, and obtain the global model from CS.

In order to further optimize the global model, users, and CS will iteratively perform steps 2) and 3) until the global model parameters meet a set convergence condition.

C. Threat Model

We assume that some users and the server may be dishonest, but TA is fully trusted who will not collude with any malicious attacker or participant in our protocol. Therefore, we consider the following threatening adversaries.

1) *Untrusted CS:* In the proposed protocol, the CS is untrusted, which means it not only may want to obtain the private information of the users from their encrypted local model parameters, but also may return wrong aggregation results to the users. During the phase of aggregation, it may collude with some malicious users to obtain the private information of honest users and generate wrong aggregation results to cheat the honest users.
2) *Malicious Users:* In the proposed protocol, malicious users may analyze the private information of honest users from their ciphertexts of local model parameters. They may also collude with the untrusted server to obtain the private information of honest users and generate wrong aggregation result to cheat the honest users. However, we will not consider the condition that malicious users upload wrong gradient parameters, such as the poisoning attack [22].

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

3) *Third-Party Attackers:* The adversaries except untrusted CS 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 secure, verifiable and traceable, and malicious attackers cannot obtain any valuable information during the training process.

D. LWE Homomorphic Encryption

An LWE problem was introduced by Regev [23]. Subsequently, Lindner and Peikert [24] proposed a new LWE-based cryptosystem. The cryptosystem involves an integer modulus $q > 2$ and integer dimensions n_{lwe}. Fix another integer p so that $gcd(p, q) = 1$. Below, $\overset{g}{\leftarrow} \mathbb{Z}_{(0,\sigma)}$ is denoted as the sampling from the Gaussian distribution with mean 0 and standard deviation σ. And, $\overset{\$}{\leftarrow}$ means sampling uniformly at random. The LWE HE can be expressed as follows.

1) *ParamGen(m):* $m \in \mathbb{Z}_p^{1\times d}$ is a plaintext vector with d dimensions. Return $\text{LWE.pp} = (q, d, p)$.
2) *KeyGen (LWE.pp):* Take uniformly random $R, S \overset{g}{\leftarrow} \mathbb{Z}_{(0,\sigma)}^{n_{\text{lwe}}\times d}$ and $A \overset{\$}{\leftarrow} \mathbb{Z}_q^{n_{\text{lwe}}\times n_{\text{lwe}}}$. Compute $Y = pR - AS \in Z_q^{n_{\text{lew}}\times d}$. Return the public key $pk = (A, Y)$, and the secret key $sk = S$.
3) *LWE.Enc (pk, m):* On input the public key pk and plaintext m, the algorithm generates the LWE ciphertext. Take $e_1, e_2 \overset{g}{\leftarrow} \mathbb{Z}_{(0,\sigma)}^{1\times n_{\text{lwe}}}$, $e_3 \overset{g}{\leftarrow} \mathbb{Z}_{(0,\sigma)}^{1\times d}$. Compute $c_1 = e_1 A + pe_2 \in \mathbb{Z}_q^{1\times n_{\text{lwe}}}$, $c_2 = e_1 Y + pe_3 + m \in \mathbb{Z}_q^{1\times d}$. Return ciphertext $c = (c_1, c_2) \in \mathbb{Z}_q^{1\times (n_{\text{lwe}}+d)}$.
4) *LWE.Dec(sk, $c = (c_1, c_2)$):* On input the secret key sk and ciphertext $c = (c_1, c_2)$, the algorithm recovers the plaintext $m = (c_1 sk + c_2) \bmod p$.

The correctness of this cryptosystem was proved in [12]. Details of the process are as follows:

$$\begin{aligned}
&(c_1 sk + c_2) \bmod p \\
&\quad = ((e_1 A + pe_2)\cdot S + e_{i1} Y + pe_3 + m) \bmod p \\
&\quad = (e_1 \cdot AS + e_1(pR - AS) + m) \bmod p \\
&\quad = (e_1 \cdot AS - e_1 \cdot AS + m) \bmod p \\
&\quad = m \bmod p.
\end{aligned}$$

E. Linear Homomorphic Hash

LHH is proposed to solve the problem of verification in the distributed system because of its homomorphism [25]. Generally, users can calculate the hash value of a single block if the message is divided into several blocks. Then, users combine these hash values and verify the blocks of message when the combination is equal to the hash value of the message. LHH consists of two algorithms as follows. Below the string g is the generator of the cyclic group $\mathbb{G}$ with prime order q. And, $g_1, \dots, g_d$ is discrete elements randomly selected in the cyclic group.

1) *LHH.Hash(x):* On input a vector x of d dimensions, the algorithm generates the linearly homomorphic hash of $x : h \leftarrow \prod_{i\in[d]} g_i^{x[i]} \in \mathbb{G}$.
2) *LHH.Eval($h_1, \dots, h_\ell, \alpha_1, \dots, \alpha_\ell$):* On input ℓ hash functions $h_1, \dots, h_\ell$ and uniformly random coefficients $\alpha_1, \dots, \alpha_\ell$ in the field $\mathbb{Z}_q$, this algorithm generates the linear combinations of ℓ hash values: $h^* \leftarrow \prod_{i\in[\ell]} h_i^{\alpha_i}$.

Note that this construction of LHH is collision resistant [9]. And, LHH satisfies linear homomorphism

$$\begin{aligned}
&\text{LHH.Hash}(\alpha_1 x_1 + \cdots + \alpha_\ell x_\ell) \\
&\quad = \prod_{i\in[d]} g_i^{\alpha_1 x_1[i]+\cdots+\alpha_\ell x_\ell[i]} \\
&\quad = \prod_{i\in[d]} g_i^{\alpha_1 x_1[i]} \times \cdots \times \prod_{i\in[d]} g_i^{\alpha_\ell x_\ell[i]} \\
&\quad = \text{LHH.Hash}(\alpha_1 x_1) \times \cdots \times \text{LHH.Hash}(\alpha_\ell x_\ell) \\
&\quad = h_1^{\alpha_1} \times \cdots \times h_i^{\alpha_i} \\
&\quad = \text{LHH.Hval}(h_1, \dots, h_\ell, \alpha_1, \dots, \alpha_\ell).
\end{aligned}$$

III. Analysis of the Previous Protocols

In this section, we analyze the verification phase in VerifyNet [8] and VeriFL [9] protocols in detail and point out their security vulnerabilities. The VerifyNet protocol cannot verify effectively if the server colludes with any user and forges the results. In the VeriFL protocol, the integrity of the LHH used for verification can not be guaranteed. The attacker can send the wrong gradient vector and LHH to the honest users by forging the commitment value. Eventually, the aggregation results of wrong gradient vector are considered real and accepted by the users.

A. Analysis of the VerifyNet Protocol

A brief introduction of the verification phase in VerifyNet is shown in Fig. 3, and please see [8] for the details of the protocol. G_1 and G_2 are multiplicative cyclic groups with the same prime order q, where q is a large prime. g and h are randomly chosen from G_1 and G_2. Both δ and ρ are selected in finite field Z_q. $HF_{\delta,\rho}(x_n)$ is a collision-resistant homomorphic hash function [26], which satisfies addition and coefficient multiplication homomorphism and takes local gradient vector x_n as input. PF_K is a pseudorandom function with secret key K [27]. $e(g, h)$ is a bilinear pairing which satisfies $e(g^a, h^b) = e(g, h)^{ab}$, where a and b are random numbers of finite field Z_q. However, such verification only applies to honest servers. The wrong aggregation results can still pass the verification if the server colludes with any malicious user. The detailed analysis is listed as follows.

Suppose the server colludes with a malicious user and replaces the real aggregation result $\sum x_n$ with a forged aggregation result σ. Then the server recalculates the following values using the secret key $\delta, \rho, K = (K_1, K_2), d$, the user n and the total number of users τ obtained from the collusive user and calculates

$$A' = g^{HF_{\delta,\rho}(\sigma)},\ B' = h^{HF_{\delta,\rho}(\sigma)} \tag{4}$$

$$L' = g^{(\varphi - HF_{\delta,\rho}(\sigma))^{1/d}} \tag{5}$$

$$Q' = h^{(\varphi - HF_{\delta,\rho}(\sigma))^{1/d}}. \tag{6}$$

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

Verification Phase of VerifyNet Protocol

- **Round 0 (Initialization)**
  TA :
  Generate a public/private key pair (N_n^{PK}, N_n^{SK}), (P_n^{PK}, P_n^{SK}) and secret kety $\{(\delta,\rho),\ K=(K_1,K_2)\}$to rach user n.
- **Round 1 (Key Sharing)**
  User n :
  The user n secretly shares the seed of **PRG** and its private key with other users.
- **Round 2 (Masking Input)**
  User n :
  The user n uses **KA**.**Agree** to generate another seed of **PRG**, so as to encrypt the local gradient vector x_n with double masking. At the same time, in order to verify the results returned from the cloud server, the user also calculates the following additional information:

$$HF(x_n)=(A_n,B_n)=(g^{HF_{\delta,\rho}(x_n)},\ h^{HF_{\delta,\rho}(x_n)}).$$
$$PF_{K_1}(n)=(\gamma_n,\ v_n);\quad PF_{K_2}(\tau)=(\gamma,\ v).$$
$$PF_K(n,\tau)=(E_n,F_n)=(g^{\gamma_n\gamma+v_nv},\ h^{\gamma_n\gamma+v_nv}).$$
$$L_n=(E_n\cdot A_n^{-1})^{1/d}=(g^{\gamma_n\gamma+v_nv-HF_{\delta,\rho}(x_n)})^{1/d}.$$
$$Q_n=(F_n\cdot B_n^{-1})^{1/d}=(h^{\gamma_n\gamma+v_nv-HF_{\delta,\rho}(x_n)})^{1/d}.$$

  where d is a selected positive integer. Finally, the user sends the ciphertext and $(A_n,\ B_n,\ L_n,\ Q_n,\ \omega_n=1)$ to the cloud server.
- **Round 3 (Unmasking)**
  Server:
  The cloud server removes the mask by restoring the shared secret of the offline user to obtain aggregation results $\sum x_n$. Then calculate:

$$A=\prod A_n;\ B=\prod B_n;\ L=\prod L_n;\ Q=\prod Q_n;\ \Omega=\prod \Omega_n$$

  and send $\{\sum x_n,\ A,\ B,\ L,\ Q,\ \Omega\}$ to the online users.
- **Round 4 (Verification)**
  User n :
  Known $PF_{K_1}(n)=(\gamma_n,\ v_n)$ and $PF_{K_2}(\tau)=(\gamma,\ v)$, calculates $\varphi=\sum(\gamma_n\gamma+v_nv)$ and $\Phi=e(g,h)^{\varphi}$.
  Verify the following equation:

$$(A,B)\overset{?}{=}(g^{HF_{\delta,\rho}(\sum x_n)},\ h^{HF_{\delta,\rho}(\sum x_n)})$$
$$e(A,h)\overset{?}{=}e(g,B)$$
$$e(L,h)\overset{?}{=}e(g,Q)$$
$$\Phi\overset{?}{=}e(A,h)\cdot e(L,h)^d.$$

  If any of the above equations are not valid, reject the aggregated result. Otherwise, accept the result.

Fig. 3. Verification phase of VerifyNet.

The malicious server can calculate $\varphi=\sum(\gamma_n\gamma+v_nv)$ easily

$$PF_{K_1}(n)=(\gamma_n,v_n)$$
$$PF_{K_2}(\tau)=(\gamma,v)$$

where K_1 and K_2 are the secret key of the pseudorandom functions PF_{K_1} and PF_{K_2}. All users are given the same secret key. So the server can calculate σ, A', B', L', and Q' and send them to the honest user.

Then, the honest user receives these values and verifies as follows:

$$(A',B')\overset{?}{=}\left(g^{HF_{\delta,\rho}(\sigma)},B'=h^{HF_{\delta,\rho}(\sigma)}\right)\tag{7}$$
$$e(A',h)\overset{?}{=}e(g,B')\tag{8}$$
$$e(L',h)\overset{?}{=}e(g,Q')\tag{9}$$
$$\Phi=e(g,h)^{\varphi}\overset{?}{=}e(A',h)\cdot e(L',h)^d.\tag{10}$$

Obviously, (7) holds because (A',B') is calculated by the server with the wrong result σ of (4). According to the properties of bilinear pairs, it can be inferred that (8) and (9) are also correct

$$e(A',h)=e\left(g^{HF_{\delta,\rho}(\sigma)},h\right)=e\left(g,h^{HF_{\delta,\rho}(\sigma)}\right)=e(g,B')$$
$$\begin{aligned}e(L',h)&=e\left(g^{(\sum(\gamma_n\gamma+v_nv)-HF_{\delta,\rho}(\sigma))},h\right)^{1/d}\\&=e\left(g,h^{(\sum(\gamma_n\gamma+v_nv)-HF_{\delta,\rho}(\sigma))}\right)^{1/d}\\&=e(g,Q').\end{aligned}$$

It is also not difficult to prove that (10) is true

$$\begin{aligned}\Phi&=e(A',h)\cdot e(L',h)^d\\&=e\left(g^{HF_{\delta,\rho}(\sigma)},h\right)\cdot e\left(g^{(\sum(\gamma_n\gamma+v_nv)-HF_{\delta,\rho}(\sigma))},h\right)\\&=e\left(g^{\sum(\gamma_n\gamma+v_nv)},h\right)\\&=e(g,h)^{\varphi}.\end{aligned}$$

Therefore, the verification of VerifyNet has a security vulnerability. Once the server colludes with a malicious user and obtains the parameters δ,ρ,K,d, and τ during the phase of verification, it can forge the aggregation results which can still pass the verification. In the following section, we will consider reducing shared parameters between users to prevent collusion and forgery by the servers and malicious users.

B. Analysis of the VeriFL Protocol

The verification phase of VeriFL is shown in Fig. 4. This protocol can verify the aggregation results and reduce the computational overhead of verification compared with other work [8], [28]. Its verification is performed after the end of all epochs, but other protocols verify the result in each epoch such as VerifyNet.

In VeriFL, users use the LHH function to verify the results, and use the commitment to prevent the hash value from being maliciously tampered with. However, the hash value and the aggregation result may be forged if the server is malicious. The detailed analysis is shown as follows:

Assume that the malicious server receives a hash value $h_j\langle e\rangle$, a commitment value $c_j\langle e\rangle$, and a random element $r_j\langle e\rangle$ from honest users P_j in eth epoch. Then, it selects a vector $v_j'\langle e\rangle$ instead of the real gradient vector and calculates the new hash value $h_j'\langle e\rangle=\text{LHH.Hash}(v_j'\langle e\rangle)$ and commitment $c_j'\langle e\rangle=\text{COM.Commit}(h_j'\langle e\rangle,\ r_j\langle e\rangle)$, and forges the aggregation results $a'\langle e\rangle=\sum_{i\in U_3\langle e\rangle,i\neq j}v_i\langle e\rangle+v_j'\langle e\rangle$. Finally, it sends $\{c_j'\langle e\rangle,\ h_j'\langle e\rangle, a'\langle e\rangle\}$ to honest users. The honest users receive these values and start the verification phase

$$\text{COM.Decommit}\left(c_j'\langle e\rangle,\ h_j'\langle e\rangle,\ r_j\langle e\rangle\right)\overset{?}{=}1\tag{11}$$
$$h^*\overset{?}{=}\text{LHH.Hash}(a^*)\tag{12}$$

where h^* is the linear combination of the hash value and a^* is the linear combination of aggregation results in different

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

Verification Phase of VeriFL Protocol

(for the epoch $k \in [\ell]$)

- **Round 0 (Commitment)**

  User $\mathrm{P_i} \in U_3\langle k\rangle$:

  The user P_i calculates a linear homomorphic hash and a commitment for the local gradient v_i $(k \in [\ell])$ and send $\{c_i\langle k\rangle,\ h_i\langle k\rangle,\ r_i\langle k\rangle\}$to the cloud server:

  $$h_i\langle k\rangle \leftarrow LHH.Hash(v_i\langle k\rangle)$$
  $$c_i\langle k\rangle \leftarrow COM.Commit(h_i\langle k\rangle; r_i\langle k\rangle)$$

  where $r_i\langle k\rangle$ is uniformly drawn from randomness space by P_i.

  Cloud Server:

  The cloud server receives the message and appends eachclient that is alive at present to the set V_1. And then server returns $\{c_j\langle k\rangle,\ h_j\langle k\rangle,\ r_j\langle k\rangle\}$ and the set V_1 to other users $P_j \in V_1$.

- **Round 1 (Decommitment)**

  User $\mathrm{P_i} \in V_1$:

  Each user P_i sends its shares of $\{h_j\langle k\rangle,\ r_j\langle k\rangle\}$ to the server,in which $j \in U_3 \backslash V_1$ (This represents users who did not successfully upload hash and commitment values to the cloud server in the **Round 0**).

  Cloud Server:

  The server reconstructs $\{h_j\langle k\rangle,\ r_j\langle k\rangle\}$ and sends them to each alive users $P_i \in V_2$.

- **Round 2 (Checking)**

  User $\mathrm{P_i} \in V_2$:

  Check that $COM.Decommit(c_j\langle k\rangle, h_j\langle k\rangle, r_j\langle k\rangle) = 1$ holds for all $k \in [\ell]$ and $j \in U_3\langle k\rangle \backslash V_1$; otherwise output $\perp$.

  Check that $h^* = LHH.Hash(a^*)$. Output the aggregation results $(a\langle 1\rangle\rangle, \ldots, a\langle \ell\rangle)$ if it holds; otherwise output $\perp$, where :

  $$h^* \leftarrow LHH.Eval(h\langle 1\rangle, \ldots, h\langle \ell\rangle, \alpha_1, \ldots, \alpha_\ell)$$
  $$a^* \leftarrow \sum_{k\in[\ell]} \alpha_k a\langle k\rangle$$

Fig. 4. Verification phase of VeriFL.

epochs. It is obvious that (11) holds, because $c'_j\langle e\rangle$ is calculated by the server

$$c'_j\langle e\rangle \leftarrow \mathrm{COM.Commit}\Big(h'_j\langle e\rangle,\ \ r_j\langle e\rangle\Big).$$

The above forgery process is equivalent to the condition that server constructs some nonexistent users and generates their hash values and commitment values like other honest users. So (12) is also true. The specific proof is as follows:

$$\begin{aligned} h^* &= \mathrm{LHH.Eval}\big(h\langle 1\rangle, \ldots, h'\langle e\rangle, \ldots, h\langle \ell\rangle, \alpha_1, \ldots, \alpha_e, \ldots, \alpha_\ell\big) \\ &= \left(\prod_{k\in[\ell],k\neq e} h\langle k\rangle^{\alpha_k}\right) h'\langle e\rangle^{\alpha_e} \\ &= \left(\prod_{k\in[\ell],k\neq e} \left(\prod_{i\in U_3\langle k\rangle} h_i\langle k\rangle\right)^{\alpha_k}\right) h'\langle e\rangle^{\alpha_e} \\ &= \prod_{j\in[d]} g_j^{\sum_{k\in[\ell],k\neq e} \alpha_k a\langle k\rangle[j] + \alpha_e a'\langle e\rangle[j]} \\ \mathrm{LHH.Hash}\big(a^*\big) &= \mathrm{LHH.Hash}\left(\sum_{k\in[\ell],k\neq e} \alpha_k a\langle k\rangle + \alpha_k a'\langle e\rangle\right) \\ &= \prod_{j\in[d]} g_j^{\sum_{k\in[\ell],k\neq e} \alpha_k a\langle k\rangle[j] + \alpha_e a'\langle e\rangle[j]} \end{aligned}$$

where $\{\alpha_1, \ldots, \alpha_e, \ldots, \alpha_\ell\}$ are random coefficents, $h'\langle e\rangle = \mathrm{LHH.Eval}(\{h'_i\langle e\rangle, h'_j\langle e\rangle\}_{i\in U_3\langle e\rangle}, 1, \ldots, 1)$, and the set $U_3\langle e\rangle$ represents the users who have real gradient vector in the

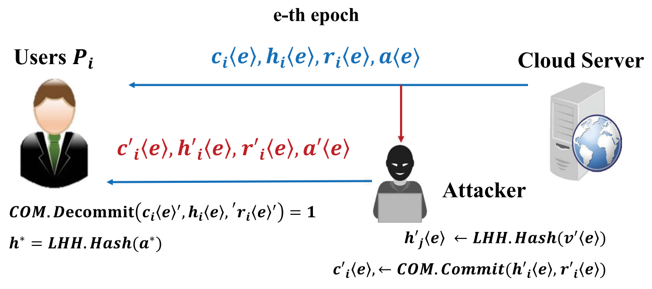
e-th epoch
Users P_i
$c_i\langle e\rangle, h_i\langle e\rangle, r_i\langle e\rangle, a\langle e\rangle$
Cloud Server
$c'_i\langle e\rangle, h'_i\langle e\rangle, r'_i\langle e\rangle, a'\langle e\rangle$
Attacker
$COM.\mathrm{Decommit}(c_i\langle e\rangle', h_i\langle e\rangle, 'r_i\langle e\rangle') = 1$
$h^* = LHH.Hash(a^*)$
$h'_j\langle e\rangle \leftarrow LHH.Hash(v'\langle e\rangle)$
$c'_i\langle e\rangle, \leftarrow COM.Commit(h'_i\langle e\rangle, r'_i\langle e\rangle)$

Fig. 5. Attack on the verification process of VeriFL protocol.

eth epoch. Honest users complete the verification, but finally receive the incorrect aggregate results.

Note that not only the server but also any attacker can forge the aggregation results. Because messages $\{c_j\langle k\rangle,\ h'_j\langle k\rangle, a\langle k\rangle\}$ are sent to the CS through insecure channel, and any attacker can easily obtain these values. The specific attack process is shown in Fig. 5, where the commitment no longer works. Any attacker can change the value $c_i\langle k\rangle$ because of the public transmission and of randomness $r_i\langle k\rangle$.

After analyzing the above two protocols, we find that the reason why two protocols cannot be verified effectively is that they disclose the important values involved in verification, and it is easy for attackers to tamper with these verification values. Hence, we need to protect these important values so that attackers cannot forge them.

IV. Our Proposed Protocol

In this section, we introduce the technical details of our proposed protocol. The purpose of this protocol is to solve several challenges in FL. First, we must protect the local gradient and the aggregated global gradient of users during the training process. Second, because the server is not fully trusted and may return wrong results to deceive users, the users need to have the ability to effectively verify the aggregation results. Third, users should have the ability of reverse tracking to determine which epoch and which user causes the failed verification process. Finally, due to multiple interactions between the users and the server in the protocol, some users may drop off, which will result in users not getting the correct aggregation results. The above situations are all considered in our protocol, and the correct results can be obtained even if some users disconnect.

A. Phase of Aggregation

The detailed description of the aggregation phase in our protocol is shown in Figs. 6 and 7, which needs five rounds to complete secure aggregation. Specifically, TA is a trusted third party. It not only generates all required public-private key pairs in the system, but also initializes the parameters of the whole training model. Then, a user P_i encrypts its local gradient v_i using double-masking and LWE-based encryption and submits it to the server. When the server receives messages sent by all online users, it starts to aggregate these ciphertexts. The server can use the secret share

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

1. **Common reference string**

   The threshold t, the number of parties N, the input domain $Z_Q^{1\times d}$, the aggregation should lie in $Z_B^{1\times d}$ for some bound $B \geq N \cdot Q$, and the epoch size ℓ. The mark $\overset{g}{\leftarrow}$ is for "sampling randomly from a discrete Gaussian distribution", so that $y \overset{g}{\leftarrow} Z_{(0,\sigma)}$ means y appears from Gaussian distribution with mean 0 and standard deviation σ. And $x \overset{\$}{\leftarrow} Z_p^n$ means x is a vector of n elements from randomly uniform distribution modulo p. Generate integers $q \in Z^+, p \in Z^+$, so that $gcd(p,q) = 1$ and $q > p > B$.

2. **Aggregation phase (for the epoch $k \in [\ell]$)**

   - **Round 0 (Generate public parameters)**
     TA:
     - Take integer $n_{lew}\langle k\rangle \in \mathbb{Z}^+$.Take $R\langle k\rangle$, $S\langle k\rangle \overset{g}{\leftarrow} \mathbb{Z}_{(0,\sigma)}^{n_{lew}\langle k\rangle\times d}$, $A\langle k\rangle \overset{\$}{\leftarrow} \mathbb{Z}_q^{n_{lew}\langle k\rangle\times n_{lew}\langle k\rangle}$.
     - Compute $Y\langle k\rangle = pR\langle k\rangle - A\langle k\rangle S\langle k\rangle \in Z_q^{n_{lew}\times d}$. Return the public key $pk = (A\langle k\rangle, Y\langle k\rangle, n_{lwe}\langle k\rangle)$, and the secret key $S\langle k\rangle$ for each user $P_i (i \in \{1,\ldots,N\})$. Generate three pairs $(sk_i\langle k\rangle, pk_i\langle k\rangle)$, $(msk_i\langle k\rangle, mpk_i\langle k\rangle)$ and $(ssk_i\langle k\rangle, spk_i\langle k\rangle)$ for each P_i, that all the public keys are different.
     - Send public keys $\{i, pk_i\langle k\rangle, mpk_i\langle k\rangle, spk_i\langle k\rangle\}_{i\in[N]}$ to all users and the server and secret key $(sk_i\langle k\rangle, msk_i\langle k\rangle, ssk_i\langle k\rangle)$ to each user.
   - **Round 1 (Secret sharing)**
     For each user P_i in parallel:
     - Receive the set $\{j, pk_j\langle k\rangle, mpk_j\langle k\rangle\}$ of other users P_j and let $U_1 \in \langle k\rangle$ be this set of users.
     - Compute the linearly homomorphic hash of its input the k-th epoch, say $v_i\langle k\rangle$, as well as the digital signature of this hash:
       $$h_i\langle k\rangle \leftarrow \mathbf{LHH}.\mathbf{Hash}(v_i\langle k\rangle), \quad sig_i\langle k\rangle \leftarrow \mathbf{Sig}.\mathbf{Gen}(h_i\langle k\rangle; ssk_i\langle k\rangle)$$
       where $\mathbf{Sig}.\mathbf{Gen}(h_i\langle k\rangle; ssk_i\langle k\rangle)$ is the signature algorithm with input of $h_i\langle k\rangle$ and signature key $ssk_i\langle k\rangle$.
     - Generate t-out-of-$(|U_1\langle k\rangle| - 1)$ shares of $h_i\langle k\rangle$: $\{[\![h_i\langle k\rangle]\!]_j\}_{j\in u_1\langle k\rangle\backslash\{i\}} \leftarrow \mathbf{SS}.\mathbf{Share}(t, u_1\langle k\rangle\backslash\{i\}, h_i\langle k\rangle)$.
     - Sample a random element $b_i\langle k\rangle \overset{\$}{\leftarrow} \mathbb{Z}_p^1$.
     - Generate t-out-of-$(|U_1\langle k\rangle| - 1)$ shares of $b_i\langle k\rangle$: $\{[\![b_i\langle k\rangle]\!]_j\}_{j\in u_1\langle k\rangle\backslash\{i\}} \leftarrow \mathbf{SS}.\mathbf{Share}(t, u_1\langle k\rangle\backslash\{i\}, b_i\langle k\rangle)$.
     - Generate t-out-of-$(|U_1\langle k\rangle|-1)$ shares of $msk_i\langle k\rangle$:$\{[\![msk_i\langle k\rangle]\!]_j\}_{j\in u_1\langle k\rangle\backslash\{i\}} \leftarrow \mathbf{SS}.\mathbf{Share}(t, u_1\langle k\rangle\backslash\{i\}, msk_i\langle k\rangle)$. Note **SS**.**Share**(threshold, number of shares, secret) is Shamir's secret sharing scheme.
     - For each other users $P_j \in U_1\langle k\rangle\backslash\{i\}$, use Deffie-Hellman key agreement protocol to compute the pairwise symmetric key $key_{i,j}\langle k\rangle \leftarrow \mathbf{KA}.\mathbf{Agree}(sk_i\langle k\rangle, pk_j\langle k\rangle)$. And calculate ciphertext using symmetric encryption algorithm $\mathbf{SK}.\mathbf{Enc}(message; key)$:
       $$C_{i,j}\langle k\rangle \leftarrow \mathbf{SK}.\mathbf{Enc}((i, j, sig_i\langle k\rangle, [\![h_i\langle k\rangle]\!]_j, [\![b_i\langle k\rangle]\!]_j, [\![msk_i\langle k\rangle]\!]_j);\ key_{i,j}\langle k\rangle).$$
   - **Round 2 (Encrypt gradient)**
     For each user P_i in parallel:
     - Take $e_{i1}\langle k\rangle, e_{i2}\langle k\rangle \overset{g}{\leftarrow} \mathbb{Z}_{(0,\sigma)}^{1\times n_{lwe}\langle k\rangle}$, $e_{i3}\langle k\rangle \overset{g}{\leftarrow} \mathbb{Z}_{(0,\sigma)}^{1\times d}$. Compute:
       $$c_{i1}\langle k\rangle = e_{i1}\langle k\rangle A\langle k\rangle + pe_{i2}\langle k\rangle \in \mathbb{Z}_q^{1\times n_{lwe}\langle k\rangle}, \quad c_{i2}\langle k\rangle = e_{i1}\langle k\rangle Y\langle k\rangle + pe_{i3}\langle k\rangle + v_i\langle k\rangle \in \mathbb{Z}_q^{1\times d\langle k\rangle}$$
       where $v_i\langle k\rangle \in \mathbb{Z}_Q^{1\times d}$ is global gradients. The ciphertext is $c_i\langle k\rangle = (c_{i1}\langle k\rangle, c_{i2}\langle k\rangle) \in \mathbb{Z}_q^{1\times(n_{lwe}\langle k\rangle+d)}$.
     - For each other $P_j \in U_1\langle k\rangle\backslash\{i\}$, compute another agreement key for double-masking:
       $$mak_{i,j} \leftarrow \mathbf{KA}.\mathbf{Agree}(msk_i\langle k\rangle, mpk_j\langle k\rangle)$$
     - Compute the double-masking of $c_i\langle k\rangle$
       $$M_i\langle k\rangle \leftarrow c_i\langle k\rangle + PRG(b_i\langle k\rangle) + \sum_{j\in U_1\langle k\rangle} \triangle_{i,j} PRG(mak_{i,j}),$$
       where $\triangle_{i,j} = 1$ if $i < j$ and $\triangle_{i,j} = -1$ if $i > j$ and $\triangle_{i,j} = 0$ if $i = j$.
     - If any of the above operations fails, abort; otherwise send $M_i\langle k\rangle$ and all tuples $(i, \{(j, C_{i,j}\langle k\rangle)\}_{j\in U_1\langle k\rangle\backslash\{i\}})$ to the server.

     Server:
     - Collect $M_i\langle k\rangle$ from at least t users. Let $U_2\langle k\rangle \subseteq U_1\langle k\rangle$ be this set of users. Send to each user $P_j \in U_2\langle k\rangle$ all messages for it: $\{(i, C_{i,j}\langle k\rangle)\}_{i\in U_2\langle k\rangle\backslash\{j\}}$ and the set $U_2\langle k\rangle$.

Fig. 6. Aggregation phase of our protocol (Round 0–Round 2).

of the offline users sent by online users to reconstruct their secrets. In the end, users get the global gradient and then return to the first round to start a new iterative process until other configurations, such as the efficiency or accuracy of the trained network model meet the constraints set of the initial goal.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

- **Round 3 (Unmasking)**
  For each party P_i in parallel:
  - Receive from the server the set of messages $\{(j, C_{i,j}\langle k\rangle)\}_{j\in U_2\langle k\rangle\setminus\{i\}}$ and a set $U_2\langle k\rangle$. Verify that $U_2\langle k\rangle \subseteq U_1\langle k\rangle$ and $|U_2\langle k\rangle| \geq t$.
    If any of above operations fails, abort.
  - For each user $P_j \in U_1\langle k\rangle\setminus\{i\}$, decrypt the ciphertext $C_{i,j}\langle k\rangle$, which was received in the Round 2. Assert $j' = j$ and $i' = i$.
    $$(j', i', sig_{i'}\langle k\rangle,\ [\![h_{j'}\langle k\rangle]\!]_i,\ [\![b_{j'}\langle k\rangle]\!]_i,\ [\![msk_{j'}\langle k\rangle]\!]_i) \leftarrow \mathbf{SK.Dec}(C_{i,j}\langle k\rangle; key_{i,j}\langle k\rangle).$$
  - Send its secret seed for self-mask and two sets of shares $(b_i\langle k\rangle, \{[\![b_j\langle k\rangle]\!]_i\}_{j\in U_2\langle k\rangle}, \{[\![msk_j\langle k\rangle]\!]_i\}_{j\in U_1\langle k\rangle\setminus U_2\langle k\rangle})$ to the server.

  Server:
  - Collect message from at least t users. Let $U_3\langle k\rangle \in U_2\langle k\rangle$ be this set of users. If $|U_3\langle k\rangle| < t$, abort.
  - For each $P_i \in U_2\langle k\rangle\setminus U_3\langle k\rangle$, reconstruct the original secret $b_i\langle k\rangle \leftarrow \mathbf{SS.Recon}(t, \{[\![b_i\langle k\rangle]\!]_j\}_{j\in U_3\langle k\rangle})$ by using Shamir's secret sharing scheme.
  - For each $P_i \in U_1\langle k\rangle\setminus U_2\langle k\rangle$, reconstruct $msk_i\langle k\rangle \leftarrow \mathbf{SS.Recon}(t, \{[\![msk_i\langle k\rangle]\!]_j\}_{j\in U_3\langle k\rangle})$. So the $msk_{i,j}\langle k\rangle$ for all $j \in U_2\langle k\rangle$.
  - Compute $\sum_{i\in U_2\langle k\rangle} c_i\langle k\rangle \leftarrow \sum_{i\in U_2\langle k\rangle} M_i\langle k\rangle - \sum_{i\in U_2\langle k\rangle} PRG(b_i\langle k\rangle) + \sum_{i\in U_1\langle k\rangle\setminus U_2\langle k\rangle, j\in U_2\langle k\rangle} \triangle_{i,j} PRG(mak_{i,j})$.
  - Send $(\sum_{i\in U_2\langle k\rangle} c_i\langle k\rangle, U_3\langle k\rangle)$ to all users in $U_3\langle k\rangle$.
- **Round 4 (Decrypt)**
  For each party P_i in parallel:
  - Receive from the server the set of messages $\{\sum_{i\in U_2\langle k\rangle} c_i\langle k\rangle, U_3\langle k\rangle\}$. Verify that $U_3\langle k\rangle \in U_2\langle k\rangle$, $|U_3\langle k\rangle| \geq t$.
    If any of above operations fails, abort.
  - Split the vector $\sum_{i\in U_2\langle k\rangle} c_i\langle k\rangle = (\sum_{i\in U_2\langle k\rangle} c_{i1}\langle k\rangle, \sum_{i\in U_2\langle k\rangle} c_{i2}\langle k\rangle) \in \mathbb{Z}^{1\times(n_{lwe}\langle k\rangle + d)}$.
  - Compute the the aggregation result $a\langle k\rangle = \left((\sum_{i\in U_2\langle k\rangle} c_{i1}\langle k\rangle S\langle k\rangle + \sum_{i\in U_2\langle k\rangle} c_{i2}\langle k\rangle)\, mod\, p\right)\, mod\, B$.

Fig. 7. Aggregation phase of our protocol (Round 3 and Round 4).

B. Phase of Verification

There are five rounds in the verification phase of our protocol, and the detailed description is shown in Fig. 8. All online users verify the results through LHH after getting the aggregation results of all epochs. And, users decide to accept or reject the aggregated results returned by the server. Note that each user has two values for verification that need to be generated in the aggregation phase and sent to the server: 1) the LHH value of user's own local gradient and 2) the digital signature of the hash value. A digital signature is used to verify the authenticity and integrity of the hash value for other users and prevent malicious attackers from forging the hash value.

C. Correctness of Our Protocol

In the following content, we analyze the correctness of the aggregation and verification in our protocol.

1) Correctness of the Aggregation:

Theorem 1: In our protocol, each user can obtain correct aggregated gradients $a = \sum_{P_i\in[N]} v_i$ to update the model if it executes the protocol honestly.

Proof: In our protocol, the gradient vector is disturbed twice by users P_i, $i \in \{1, \ldots, N\}$. It is first encrypted by the LWE algorithm, and then disturbed by the double-masking technique. The local gradient vectors v_i are encrypted by using LWE as follows:

$$\begin{aligned} c_i &= (c_{i1}, c_{i2}) \\ &= (e_{i1}A + pe_{i2},\ e_{i1}Y + pe_{i3} + v_i) \end{aligned} \tag{13}$$

and disturbed again by using the double-masking method as

$$M_i = c_i + \mathrm{PRG}(b_i) + \sum \Delta_{i,j}\mathrm{PRG}\big(\mathrm{mak}_{i,j}\big) \tag{14}$$

where $\Delta_{i,j} = -1$ if $i > j$ and $\Delta_{i,j} = 1$ if $i < j$ and $\Delta_{i,j} = 0$ if $i = j$. Then, server calculates: $\sum_{i\in U_2} c_i$ secrets by removing masks [6] and returns it to users. Finally, users decrypt it to get the aggregation result as: $a = \sum_{P_i\in[N]} v_i$. The decryption process is as follows:

$$\begin{aligned} a &= \left(\sum_{P_i\in[N]} c_{i1}S + \sum_{P_i\in[N]} c_{i2}\right) \bmod p \\ &= \left(\left(\sum_{P_i\in[N]} e_{i1}\right)AS + p\left(\sum_{P_i\in[N]} e_{i2}\right)S + \left(\sum_{P_i\in[N]} e_{i1}\right)Y \right. \\ &\qquad \left. + p\sum_{P_i\in[N]} e_{i3} + \sum_{P_i\in[N]} v_i\right) \bmod p \end{aligned} \tag{15}$$

where $Y = pR - AS$, and (15) is modified as follows:

$$\begin{aligned} a &= \left(\left(\sum_{P_i\in[N]} e_{i1}\right)AS + \left(\sum_{P_i\in[N]} e_{i1}\right)(pR - AS) \right. \\ &\qquad \left. + p\left(\sum_{P_i\in[N]} e_{i2}\right)S + p\sum_{P_i\in[N]} e_{i3} + \sum_{P_i\in[N]} v_i\right) \bmod p \\ &= \sum_{P_i\in[N]} v_i. \end{aligned} \tag{16}$$

■

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

1. **Verification phase (for the epoch $k \in [\ell]$)**
   - **Round 0 (Broadcasting)**
     For each user P_i in parallel:
     - Send $(h_i\langle k\rangle)$ to the server.

     Server:
     - Collect messages from at least t users. Let $V_1 \subseteq U_3\langle\ell\rangle$ be this set of users (note that $U_3\langle\ell\rangle$ is the latest set of surviving parties in this epoch of size ℓ). If $|V_1| < t$, abort; otherwise send the set $\{i, \{h_i\langle k\rangle_{k\in[\ell]}\}_{i\in V_1}$ to all users in V_1.
   - **Round 1 (Reconstruct Hash)**
     For each user P_i in parallel:
     - Receive from the server the set $\{i, \{h_i\langle k\rangle\}_{k\in[\ell]}\}_{i\in V_1}$ and infer the set V_1. If $|V_1| < t$, abort.
     - Send the set of shares to the server: $\{[\![h_j\langle k\rangle]\!]_i\}_{k\in[\ell], j\in U_2\langle k\rangle\setminus V_1}$.

     Server:
     - Collect sets of shares from at least t parties. Let $V_2 \subseteq V_1$be this set of parties. If $|V_2| < t$, abort.
     - Reconstruct $h_i\langle k\rangle \leftarrow$ **SS**.**Recon**$(t, \{[\![h_i\langle k\rangle]\!]_j\}_{k\in[\ell], j\in V_2})$ for each $k \in [\ell]$ and $P_i \in U_2\langle k\rangle\setminus V_1$.
     - Send $\{h_i\langle k\rangle\}_{k\in[\ell], i\in U_2\langle k\rangle\setminus V_1}$ to all users in V_2.
   - **Round 2 (Verify)**
     For each user P_i in parallel:
     - Receive $\{h_j\langle k\rangle\}_{k\in[\ell], j\in U_2\langle k\rangle\setminus V_1}$ from the server. Check whether the signature $sig_j\langle k\rangle$ is valid. Take linear homomorphic hash $h_j\langle k\rangle$and signature $sig_j\langle k\rangle$ and public key $spk_j\langle k\rangle$ as the input of the algorithm **Sig**.**Ver**$(h_j\langle k\rangle, sig_j\langle k\rangle, spk_j\langle k\rangle)$.
       The signature is valid if the output of this algorithm is true; otherwise the signature is invalid and user P_i output $\perp$.
     - For all $k \in [\ell]$ combine the linearly homomorphic hashes of different parties in the k-th epoch:
       $$h\langle k\rangle \leftarrow LHH.Eval(\{h_j\langle k\rangle\}_{j\in U_2\langle k\rangle}, 1, \ldots, 1)$$
     - Draw ℓ uniformly random coefficients in the field $\alpha_1, \ldots, \alpha_\ell \in \mathbb{Z}_{pr}$ (pr is the prime order of cyclic group $\mathbb{G}$) and compute the linear combination of combined hashes of different epoch:
       $$h^* \leftarrow LHH.Eval(h\langle 1\rangle, \ldots, h\langle\ell\rangle, \alpha_1, \ldots, \alpha_\ell).$$
     - Compute the linear combination of the aggregation result of different epochs using the same coefficients:
       $$a^* \leftarrow \sum_{k\in[\ell]} \alpha_k a\langle k\rangle \ mod\ B.$$
     - Check that $h^* = LHH.Hash(a^*)$. Output $(a\langle 1\rangle), \ldots, a\langle\ell\rangle)$ if it holds; otherwise output $\perp$.

Fig. 8. Verification phase of our protocol.

2) Correctness of the Verification:

Theorem 2: It is correct and feasible that we use LHH in [9] to verify aggregation results. Once the attacker maliciously tampered with the data, the user would fail to verify and reject the aggregation results.

Proof: The user verifies the aggregation by using the following formula:

$$h^* \stackrel{?}{=} \text{LHH.Hash}(a^*) \tag{17}$$

$$\begin{aligned} h^* &= \prod_{k\in[\ell]} h\langle k\rangle^{\alpha_k} = \prod_{k\in[\ell]}\left(\prod_{j\in U_2\langle k\rangle} h_j\langle k\rangle\right)^{\alpha_k} \\ &= \prod_{k\in[\ell]}\prod_{j\in U_2\langle k\rangle}\left(\prod_{y\in[d]} g_y^{v_j\langle k\rangle[y]}\right)^{\alpha_k} \\ &= \prod_{y\in[d]} g_y^{\sum_{k\in[\ell]}\alpha_k\left(\sum_{j\in U_2\langle k\rangle} v_j\langle k\rangle[y]\right)} \end{aligned} \tag{18}$$

$$\begin{aligned} \text{LHH.Hash}(a^*) &= \text{LHH.Hash}\left(\sum_{k\in[\ell]}\alpha_k a\langle k\rangle\right) \\ &= \prod_{y\in[d]} g_y^{\left(\sum_{k\in[\ell]}\alpha_k a\langle k\rangle[y]\right)} \end{aligned} \tag{19}$$

where $a\langle k\rangle[y]$ is the yth element in vector $a\langle k\rangle$.

If the (17) is true in the verification phase, it indicates that the aggregation result is correct, that is

$$a\langle k\rangle = \sum_{j\in U_2\langle k\rangle} v_j\langle k\rangle. \tag{20}$$

■

V. Analysis of the Aggregation Protocol

We analyze our proposed aggregation protocol, and prove it is secure and verifiable in this section. In detail, we analyze that honest users' privacy is guaranteed under different threat

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

models, and the aggregation results are verifiable. Once the aggregation result is forged, the user can detect it successfully. Then we compare our protocol with the privacy-preserving FL protocols in recent years, which shows that our protocol has obvious security advantages.

A. Security Analysis

We consider our protocol to be secure multiparty computation, and even though the CS S colludes with $t-1$ users to get the most aggressive features, they still cannot obtain the local gradients of honest users, where t represents threshold value of Shamir's secret sharing scheme. At the same time, we use the LWE encryption protocol to protect the final aggregation result. In the VeriFL protocol, although the privacy of local gradients can be protected, the security of the aggregation result cannot be ensured. In particular, during the implementation of the secret sharing protocol, users can quit at any time during the interaction between the CS and a set of U of N users. Therefore, subset U_1, U_2, U_3, V_1, V_2, and U are used to represent the surviving user set in the corresponding running wheel. And, the relations among these subsets: $V_1 \subseteq V_2 \subseteq U_3 \subseteq U_2 \subseteq U_1 \subseteq U$. For simplicity, the threshold for dropout is t, and input of user P_i is v_i. Given the threshold of secret sharing t and security parameter λ, the joint view of all parties in C can be denoted as $\text{Real}_C^{U,t,\lambda}(\{v_i\}_{i\in U}, U_3, U_2, U_1, V_2, V_1)$, where $C \subseteq U \cap \{S\}$. Therefore, we will use the following theorems to illustrate how our protocol protects the privacy of the local gradient and global gradient.

Theorem 3 (Local Model Privacy): Our protocol protects against joint attacks from multiple users and the server, assume there are threshold t, security parameter λ, a set of $|U| > t$ users, $|C\backslash\{S\}| < t$, and subsets U_1, U_2, U_3, V_1, and V_2, there exists a PPT simulator $\mathbf{SIM}_C^{U,t,\lambda}(\{v_i\}_{i\in C}, Z, U_3, U_2, U_1, V_2, V_1)$ whose output is indistinguishable from the output of $\mathbf{Real}_C^{U,t,\lambda}(\{v_i\}_{i\in U}, U_3, U_2, U_1, V_2, V_1)$, where

$$Z = \begin{cases} \sum_{i\in U_2\backslash C} v_i, & |U_2| \geq t \\ \perp, & \text{otherwise.} \end{cases} \tag{21}$$

Proof: In order to make the simulated view $\mathbf{SIM}_C^{U,t,\lambda}$ and the real view $\mathbf{Real}_C^{U,t,\lambda}$ indistinguishable, we use the simulator **SIM** to modify some parts of our protocol. The simulation of the simulator is as follows if $Z = \perp$.

In **Round 1**, simulator **SIM** replaces honest users P_i and P_j who randomly choose an encryption key $y_{i,j}$ instead of using **KA**.**Agree**. The Diffie–Hellman assumption [29] ensures that the agreement is indistinguishable from the actual agreement. Then, all the data of honest users (in the U_1) who need secret sharing are replaced by random values generated by the simulator. Such emulation does not prevent honest users from sending the correct secret sharing shares to the CS in **Round 3**. Finally, $\mathbf{PRG}(b_i)$ can also be replaced by random values rand_i generated by the simulator, so the security of **PRG** ensures that the agreement is indistinguishable from the actual agreement. Therefore, the double masking is generated as follows:

$$M_i \leftarrow \text{rand}_i + \sum_{j\in U_1\langle k\rangle} \triangle_{i,j}\text{PRG}\big(\text{mak}_{i,j}\big)$$

instead of sending

$$M_i \leftarrow c_i + \text{PRG}(b_i) + \sum_{j\in U_1} \triangle_{i,j}\text{PRG}\big(\text{mak}_{i,j}\big).$$

Since in the previous simulation, $\mathbf{PRG}(b_i)$ has been replaced by a random value and independent of any other value, obviously $\mathbf{PRG}(b_i) + c_i$ is also random. Hence, the distribution of rand_i and $\mathbf{PRG}(b_i) + c_i$ is indistinguishable. If $Z \neq \perp$ the simulation of the simulator is as follows:

Similar to the above simulation, the simulator selects a random number as the shared key between users P_i and P_j (all in the set U_2), which will be the seed of **PRG**. Specifically, a value $\text{mak}'_{i',j'}$ (P'_j is a specific user) is sampled uniformly at random for each user $P_i \in U_2\backslash C\backslash\{P'_j\}$, then **SIM** sends

$$\begin{aligned} M'_i \leftarrow c_i + \text{rand}_i + &\sum_{j\in U_1\backslash\{j'\}:i<j} \text{PRG}\big(\text{mak}_{i,j}\big) \\ &- \sum_{j\in U_1\backslash\{j'\}:i>j} \text{PRG}\big(\text{mak}_{i,j}\big) + \triangle_{i,j'}\text{PRG}\Big(\text{mak}'_{i,j'}\Big) \end{aligned}$$

where

$$\triangle_{i,j'} = \begin{cases} 1, & i < j' \\ -1, & i > j'. \end{cases}$$

Clearly, the Diffie–Hellman assumption [29] makes that the simulation process is indistinguishable from the real training process. For a Pseudorandom generator and all user $P_i \in U_2\backslash C$, the simulator can also randomly select a value $\text{rand}_{i,j}$ to replace the output of the **PRG**, like

$$M'_i \leftarrow \gamma_i + \text{rand}_i + \sum_{j\in U_1\backslash U_2\backslash C} \triangle_{i,j}\ \text{rand}_{i,j}$$

where $\{\gamma_i\}_{P_i\in U_2\backslash C}$ is randomly selected by the simulator subject to $\sum_{P_i\in U_2\backslash C} \gamma_i = \sum_{P_i\in U_2\backslash C} c_i = Z$. Thus, **SIM** can successfully simulate **Real** without knowing c_i of all parties $P_i \in C$. So through the above simulation process, we can infer that the actual output and the simulated output are indistinguishable. The proof is done. ■

Theorem 4 (Global Model Privacy): Our protocol can protect the global model privacy based on LWE-based HE [12], i.e., no useful information that the server can learn from aggregation results $\sum_{i\in U_2} v_i$.

Proof: We use a standard hybrid argument [6] to prove this theorem. Given the security integer of LWE-based HE q and n_{lwe}, we use $\text{Real}_S^{q,n_{\text{lwe}}}$ to denote the view of the server S. Then, we need to prove that there exists a PPT simulator **SIM** whose output is indistinguishable from the output of $\mathbf{Real}_S^{q,n_{\text{lwe}}}$

$$\mathbf{Real}_S^{q,n_{\text{lwe}}} \equiv \mathbf{SIM}_S^{q,n_{\text{lwe}}}. \tag{22}$$

To prove (23), we will simulate a series of random variables through the simulator **SIM**. The specific hybrid process is as follows.

Hyb$_0$: In a real execution of the protocol, we first initialize a series of random variables with the same distribution as the view of the server $\mathbf{Real}_S^{q,n_{\text{lwe}}}$.

Hyb1: In this hybrid, we use the simulator to change the behavior of each user P_i, i.e., each user P_i encrypts random variables β_i instead of the original gradient vector v_i. So the value of each user encrypted by LWE encryption is $c_i{}'$ which is not the real ciphertext c_i

$$c_i{}' = (e_1A + pe_2,\ e_1Y + pe_3 + \beta_i)$$
$$c_i = (e_1A + pe_2,\ e_1Y + pe_3 + v_i)$$

where q and n_{lwe} are the integers for security, R, $S \overset{g}{\leftarrow} \mathbb{Z}_{(0,\sigma)}^{n_{\text{lwe}}\times d}$, $e_1, e_2 \overset{g}{\leftarrow} \mathbb{Z}_{(0,\sigma)}^{1\times n_{\text{lwe}}}$, $e_3 \overset{g}{\leftarrow} \mathbb{Z}_{(0,\sigma)}^{1\times d}$, and $Y = pR - AS \in Z_q^{n_{\text{lew}}\times d}$. Y is also uniformly random, because A, R, and S are sampled uniformly at random. Then, the IND-CPA security property of LWE-based HE [12] guarantees that $c_i{}'$ and c_i have the same distribution. Thus, this hybrid is indistinguishable from Hyb0.

Hyb2: In this hybrid, we continue to change the behavior of each user. P_i adds the double mask to $c_i{}'$ as follows:

$$M_i{}' \leftarrow c_i{}' + \text{PRG}(b_i) + \sum_{j\in U_1} \Delta_{i,j}\text{PRG}\big(\text{mak}_{i,j}\big).$$

Then, each user P_i sends the simulated value $M_i{}'$ to the server S instead of the real value M_i. Reviewing the proof of Theorem 3, the local model privacy guarantees that $M_i{}'$ is computationally indistinguishable from M_i. Thus, this hybrid is indistinguishable from Hyb1.

Hyb3: In this hybrid, the server computes $\sum_{i\in U_2} c_i{}'$ in the **SIM**, and computes $\sum_{i\in U_2} c_i$ in real protocol as follows:

$$\sum_{i\in U_2} c_i{}' = \sum_{i\in U_2} M_i{}' - \epsilon$$
$$\sum_{i\in U_2} c_i = \sum_{i\in U_2} M_i - \epsilon$$

where

$$\epsilon = \sum_{i\in U_2} \text{PRG}(b_i) + \sum_{\substack{i\in U_1\setminus U_2\\ j\in U_2}} \Delta_{i,j}\text{PRG}\big(\text{mak}_{i,j}\big)$$

$\sum_{i\in U_2} c_i$ is computationally indistinguishable from $\sum_{i\in U_2} c_i{}'$, since $M_i{}'$ is exactly the same as M_i. Besides, the security of **PRG** ensures this hybrid is indistinguishable from Hyb2.

As discussed above, we prove that there exists a PPT simulator **SIM** whose output is indistinguishable from the output of $\textbf{Real}_S^{q,n_{\text{lwe}}}$. Based on security of LWE-based HE, the server learns nothing from $\sum_{i\in U_2} c_i$. Only the users can obtain the aggregation result through the LWE decryption. Hence, our protocol can realize the global model privacy. ■

Theorem 5 (Traceable Verification): Our protocol can achieve traceable verification, which means the users can not only verify the aggregation results, but also be able to identify the wrong epochs if the results are wrong.

Proof: To prove the theorem, we mainly consider the following two cases: 1) the attacker tampers with the aggregation results and the hash values and 2) the hash values are correct, but the server makes some mistakes during the aggregation phase.

In the first case, assume that the attacker tampers with the hash value and aggregation result of user P_j in the eth epoch as $h_j'\langle e\rangle$ and $a'\langle e\rangle$. So that the user P_i receives $h_j'\langle e\rangle$ and checks whether the signature is valid by the algorithm $\textbf{Sig.Ver}(h_j'\langle e\rangle, \text{sig}_j\langle e\rangle, \text{spk}_j\langle e\rangle)$. Then, the output of this algorithm is false because of the unforgeability of digital signatures. Therefore, the user can learn that the hash value of the user P_j is tampered with in the eth epoch.

In the second case, assume that the server incorrectly calculates the aggregation result as $a'\langle e\rangle$ in eth epoch. Then, the users check the validity of the signature. The signature of each user in each epoch is valid if the hash values are not tampered with. However, the equation: $h^* \overset{?}{=} \text{LHH.Hash}(a^*)$ is not true (More details are shown in the proof of Theorem 2). Meanwhile, the users can check whether the hash values are equal to those of the aggregation results in each epoch to identify the wrong epoch by the following formulas:

$$\begin{aligned} h\langle 1\rangle &\overset{?}{=} \text{LHH.Hash}\big(a'\langle 1\rangle\big)\\ &\cdots\\ h\langle e\rangle &\overset{?}{=} \text{LHH.Hash}\big(a'\langle e\rangle\big)\\ &\cdots\\ h\langle \ell\rangle &\overset{?}{=} \text{LHH.Hash}\big(a'\langle \ell\rangle\big). \end{aligned} \tag{23}$$

Take the eth epoch as an example, the derivation process is as follows:

$$\begin{aligned} h\langle e\rangle &= \prod_{j\in U_2\langle e\rangle} h_j\langle e\rangle\\ &= \prod_{j\in U_2\langle e\rangle}\left(\prod_{y\in[d]} g_y^{v_j\langle e\rangle[y]}\right)\\ &= \prod_{y\in[d]} g_y^{\sum_{j\in U_2\langle e\rangle} v_j\langle e\rangle[y]}\\ &\neq \text{LHH.Hash}\big(a'\langle e\rangle\big). \end{aligned}$$

The aggregation result is incorrect if the equation does not hold, which means that the result $a'\langle e\rangle$ returned by the server is not the real aggregation result of all users in the eth epoch. Thus, our protocol can achieve traceable verification and identify the wrong epoch through (24). ■

B. Efficiency Analysis

In the proposed protocol, we use the LWE encryption protocol to protect the gradient after aggregation, and the increase of computational overhead is tolerable as shown in Section VI. The reason is that the privacy of aggregation results is realized at the cost of appending little computational overheads of users.

Assume that the computational complexity of each call of LHH is $O(d)$. Since users need to run LHH twice to generate the hash value and verify the result. The complexity of verification is $O(d+d)$ if users verify the aggregation results in each epoch. However, the computational complexity in each epoch

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON OF SEVERAL AGGREGATION PROTOCOLS

Protocol	Privacy		Verifiability		Efficiency	
	Local Data	Aggregation Result	Verifiability	Traceability	Interaction Times	Verification Complexity
TP Le et al.[12]	×	×	×	×	1	/
Bonawitz, K et al.[6]	√	×	×	×	4	/
VerifyNet [8]	√	×	×	×	4	$O(2d)$
VeriFL [9]	√	×	√	×	4	$O(2d + d/\ell)$
Our protocol	√	√	√	√	2	$O(2d + d/\ell)$

is reduced to $O(d + d/\ell)$, if the user verifies the LHH.Eval after ℓ training epochs. Because users only run LHH.Eval once and the cost of verification is amortized by ℓ. Therefore, the total complexity of our protocol is reduced compared with VerifyNet, and equivalent to that of the VeriFL protocol as shown in Table I.

C. Comparison of Aggregation Protocols

Table I lists the comparison of several FL protocols that can achieve privacy security and verifiability, where d and ℓ represent the number of gradient vectors and the total epochs, respectively. Phong *et al.* [12] proposed a privacy-preserving FL system based on additive HE. For honest but curious servers, gradient of other users can be protected, because they are encrypted and then uploaded to the server. However, the malicious user and server may collude and obtain privacy of users. Then Bonawitz *et al.* [6] proposed a multiparty secure computation protocol, where the communication overheads of participants are low, and only one server is required. The protocol can protect the privacy of users, but the aggregation result returned by dishonest servers cannot be verified. Once the server forges the results, the global network model will be affected. In VeriFL [8] and VeriNet [9] protocols, users can be allowed to verify the correctness of the aggregation results returned by the server on the premise of acceptable overheads. Compared to VeriNet, VeriFL greatly reduces the computational costs of users in the verification phase by allocating the overheads of verification to each round. However, as shown in Section III, both protocols have some security risks that the aggregation results may be forged and the verification may not be effective.

As shown in Table I, our protocol can protect both local data and aggregated results to enhance users' privacy. In terms of verification, our protocol can verify the correctness of the results, even if the attacker forged the results. Moreover, it can also achieve traceability where the previous ones cannot find the wrong epochs effectively. In terms of efficiency, the LHH is used to improve the verification efficiency, and the number of interactions is reduced to two rounds to improve the communication efficiency.

VI. EXPERIMENTS

In this section, we mainly introduce the experimental environment and show the computational and

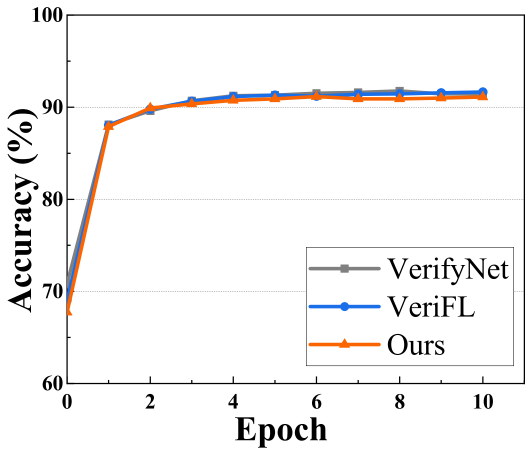

Fig. 9. Comparison of classification accuracy of three protocols.

communicational overheads of the users and server under different situations. The users and server are all simulated by 64-bit Windows10 desktop equipped with Intel AMD Ryzen 5700G (3.60 GHz) and 32-GB RAM. The experiment is single threaded. Since the end-to-end communication time can be computed in terms of bandwidth and does not significantly affect the computational complexity, we ignore it in our experiments. Our prototype is written in Python with mature libraries, which is more convenient to realize the training of neural network based on FL. We instantiate our algorithm as follows: linearly homomorphic hash bases on the NIST P-256 curve, Shamir secret sharing scheme, Diffie–Hellman key agreement, and symmetrical encryption with AES-OFB mode. Besides, we take $n_{\text{lwe}} = 3000$, $p = 2^{46} + 1$, $q = 2^{64}$ and set the maximum number of users to 1000.

A. Classification Accuracy

In order to verify the classification accuracy of the neural network model trained by this protocol, we select data from the MNIST database. At the same time, these data sets are put into the CNN network for training [30].

We implement the VerifyNet [8] and VeriFL protocols [9] in order to analyze the accuracy of our protocol on neural network training. As shown in Fig. 9, the accuracy of the VerifyNet and VeriFL protocols is almost the same, because they all use the same security aggregation scheme [6]. In practical application, the element of gradient vector is usually a

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

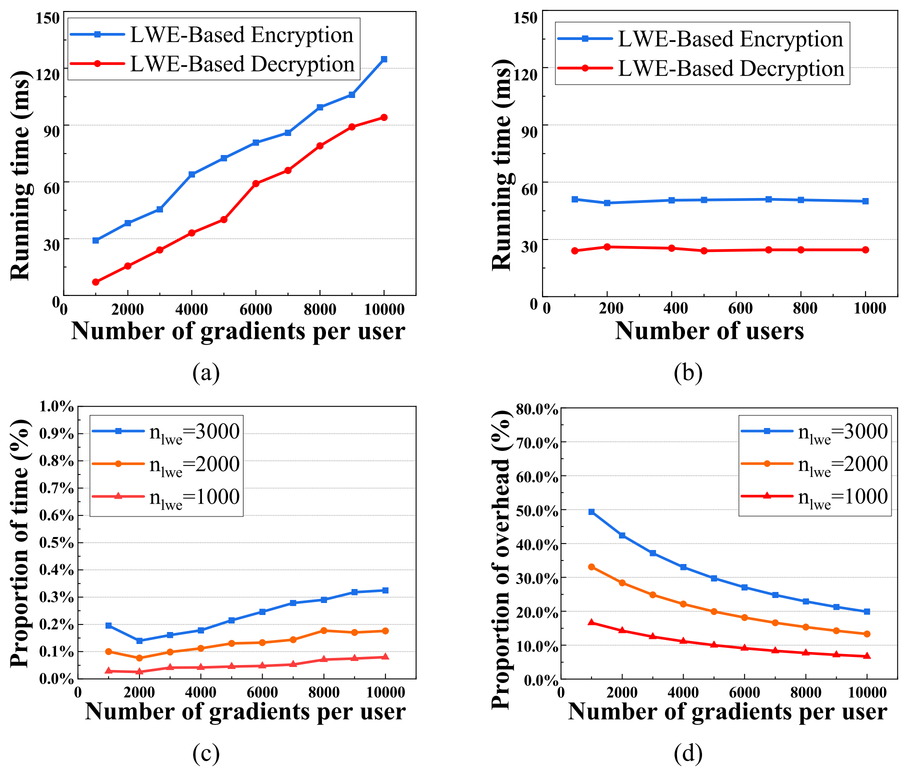

Fig. 10. (a) LWE running time with different numbers of gradients ($N = 200$). (b) LWE running time with different numbers of users ($d = 200$). (c) Proportion of LWE computing time in all of users' running time. (d) Proportion of LWE communicational overhead in all of users' communicational overheads.

floating point number, which needs to be preprocessed into integers before encryption in our protocol. This is the reason why the accuracy of our scheme was slightly lower than those of other schemes. The preprocessing method is to multiply the vector by a large integer, and finally divide the aggregation result by this integer. The neural network training model can achieve almost the same training effect by such preprocessing, which shows that the deviations caused by the preprocessing are acceptable.

B. Results of Aggregation Phase

As shown in Fig. 10(a) and (b), the running time of LWE encryption and decryption will be appended with the increase of the number of gradients, regardless of the number of users. In addition, the running time and communication overhead of LWE are also related to the security parameter n_{lwe}, because the number of the ciphertext vector is $n_{\text{lwe}} + d$, where d is the number of gradients. Fig. 10(c) and (d) record the proportion of computational and communicational overheads in the total overheads of users by using the LWE encryption scheme. The proportion of additional time will increase slowly (a large security parameter increases security but reduces efficiency), but it is far less than 1% when the number of gradients increases. On the other hand, the proportion of additional communicational overhead is reduced from 50% to 25% or less with the increase of gradients. It means that our protocol can achieve the privacy of aggregation results with acceptable communicational overhead.

Next, we change the parameters of the protocol to test the overheads of the users and the server, respectively.

1) Number of Users: As shown in Fig. 11, we show the overheads of users and server where the number of gradient for each user remains unchanged and different lines represent different dropout rates. The overheads of the user and server grow with the increase of the number of gradients. As the increase of the users, more time is needed to make multiple key agreements and generate more secret sharing, as well as more bandwidth is needed to transmit the information in Fig. 11(a) and (c). The server needs more time to remove the double-masking caused by a large number of users and recover the share of dropped users, and more communication overhead to send the results to all online users in Fig. 11(b) and (d).

2) Number of Gradients: When the number of users is fixed, the increase of gradients will lead to more calculation time of users as shown in Fig. 12(a). However, the number of gradients has little effect on the computational time of the server as shown in Fig. 12(b). Because the main computational cost of the server is to recover the secrets of dropped users, and this process is only related to the number of dropped users rather than the number of gradients.

3) Dropout Rate: As shown in Figs. 11 and 12, it is obvious that in addition to the number of users and gradients, the drop rate also has an important influence on the overhead of the server, but it has nothing to do with those of the users. The reason is that the server needs to recover the messages of dropped users, which results in unavoidable overhead, and the additional overhead is appended as the increase of dropout rate.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

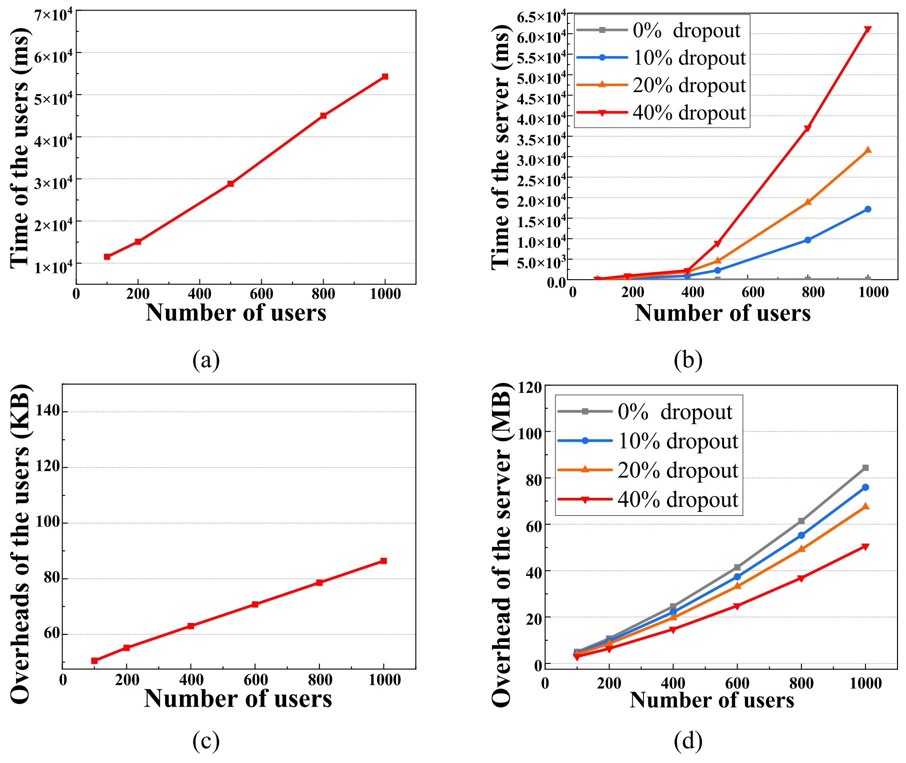

Fig. 11. Overheads in the aggregation phase with different numbers of users and dropout rate when $d = 3000$. (a) Running time of user. (b) Running time of the server. (c) Communicational overheads of the users. (d) Communicational overhead of the server.

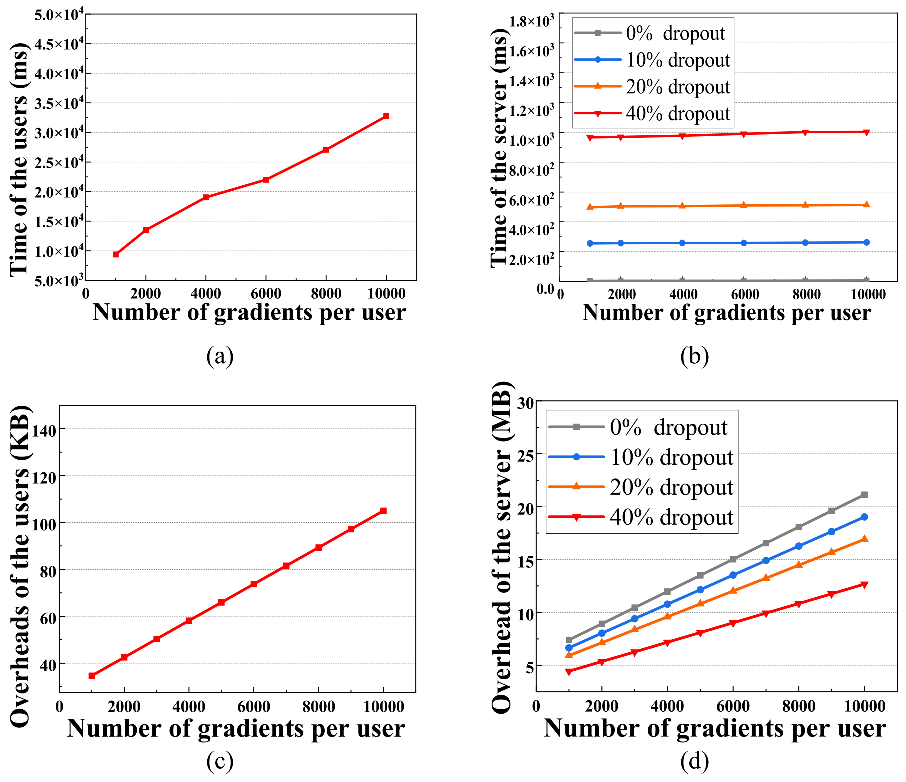

Fig. 12. Overheads in the aggregation phase with different numbers of gradients per user and dropout rate when $N = 200$. (a) Running time of user. (b) Running time of the server. (c) Communicational overheads of the users. (d) Communicational overhead of the server.

C. Results of Verification Phase

In Fig. 13, we specifically record the overheads of users and the server in the verification phase, in which the number of training epochs is 10. Here, we only consider the users who drop the line in **Round 0**, that is, they failed to send h_i to the server. In this phase, the costs of both users and the server are

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

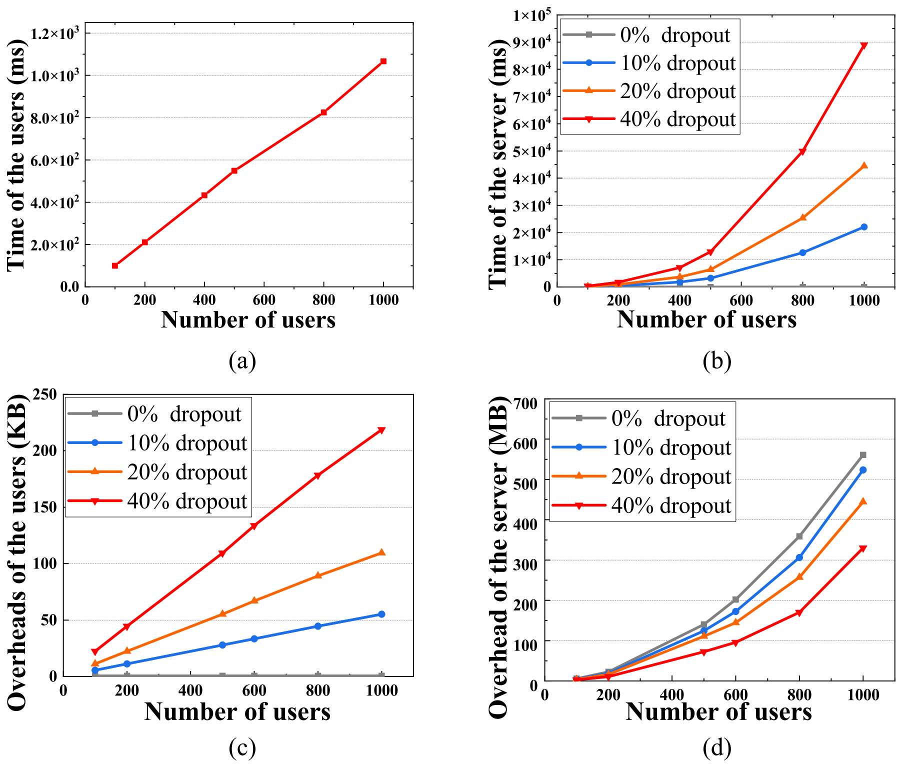

Fig. 13. Overheads in the verification phase with different numbers of users and dropout rate when $d = 5000$. (a) Running time of the users. (b) Running time of the server. (c) Communicational overheads of the users. (d) Communicational overhead of the server.

independent of the number of gradients, because the verification of LHH is needed and the gradient of any dimension can be converted to a value of fixed length. However, the number of users and dropout rate has a great impact on the overheads. As shown in Fig. 13, it is obvious that at the same dropout rate, when the number of users increases from 500 to 1000, the calculation time of users is appended about twice, and that of the server is appended about seven times. The reason is that if more users outline, the server will return (i, h_i) to fewer online users in **Round 0**. On the contrary, more algorithms of secret sharing recovery are needed to be executed and more recovered hash values are sent in **Round 1**. In addition, the communication overhead of the server in Fig. 13(d) refer to the total overhead sent to each online user. That is, assuming that the overhead of transmitting to a user is 50 kB, and that of transmitting to 100 users is $50 \times 100 = 5000$ kB. The total amount of data that the server needs to send becomes larger when the number of online users increases. Therefore, the communication overhead of the server is increased.

VII. Conclusion

In this article, we first analyze VerifyNet and VeriFL protocols and show that the verification is invalid if the server colludes with a malicious user. Further, we propose a new protocol that can effectively verify the aggregation result and achieve verification traceability even if the server colludes with malicious users. Our protocol also improves security of the FL protocol, which means that the global model, as well as the local model of users are protected. Compared with the previous ones, our protocol greatly improves security at the same level of efficiency. Finally, the experimental results show that our proposed protocol does not reduce the accuracy rate of the trained model but improves privacy security and realizes traceable verifiability. In future research, we will consider the condition that some malicious users may also execute poison attacks during the whole training process.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Stat.*, 2017, pp. 1273–1282.

[2] M. Aledhari, R. Razzak, R. M. Parizi, and F. Saeed, "Federated learning: A survey on enabling technologies, protocols, and applications," *IEEE Access*, vol. 8, pp. 140699–140725, 2020.

[3] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 603–618.

[4] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Security Privacy*, 2017, pp. 19–38.

[5] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, "HybridAlpha: An efficient approach for privacy-preserving federated learning," in *Proc. 12th ACM Workshop Artif. Intell. Security*, 2019, pp. 13–23.

[6] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 1175–1191.

[7] J. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly)logarithmic overhead," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2020, pp. 1253–1269.

[8] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and verifiable federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 911–926, 2019.

[9] X. Guo *et al.*, "VeriFL: Communication-efficient and fast verifiable aggregation for federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 1736–1751, 2020.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.

[10] X. Sun, P. Zhang, J. K. Liu, J. Yu, and W. Xie, "Private machine learning classification based on fully homomorphic encryption," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 352–364, Apr.–Jun. 2020.

[11] Q. Zhang, C. Xin, and H. Wu, "GALA: Greedy computation for linear algebra in privacy-preserved neural networks," 2021, *arXiv:2105.01827.*

[12] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Security*, vol. 13, pp. 1333–1345, 2018.

[13] D. Boneh *et al.*, "Threshold cryptosystems from threshold fully homomorphic encryption," in *Proc. Annu. Int. Cryptol. Conf.*, 2018, pp. 565–596.

[14] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu, "Efficient and privacy-enhanced federated learning for industrial artificial intelligence," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6532–6542, Oct. 2020.

[15] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proc. Theory Cryptogr. Conf.*, 2006, pp. 265–284.

[16] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 308–318.

[17] R. Hu, Y. Guo, H. Li, Q. Pei, and Y. Gong, "Personalized federated learning with differential privacy," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9530–9539, Oct. 2020.

[18] K. Wei *et al.*, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, 2020.

[19] M. A. P. Chamikara, P. Bertok, I. Khalil, D. Liu, and S. Camtepe, "Privacy preserving distributed machine learning with federated learning," *Comput. Commun.*, vol. 171, pp. 112–125, Apr. 2021.

[20] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Security Symp.*, 2016, pp. 601–618.

[21] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. 19th Int. Conf. Comput. Stat.*, 2010, pp. 177–186.

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

[23] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, pp. 1–40, 2009.

[24] R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption," in *Proc. Cryptograph. Track RSA Conf.*, 2011, pp. 319–339.

[25] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography: The case of hashing and signing," in *Proc. Annu. Int. Cryptol. Conf.*, 1994, pp. 216–233.

[26] A. Yun, J. H. Cheon, and Y. Kim, "On homomorphic signatures for network coding," *IEEE Trans. Comput.*, vol. 59, no. 9, pp. 1295–1296, Sep. 2010.

[27] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2014, pp. 844–855.

[28] A. Fu, X. Zhang, N. Xiong, Y. Gao, H. Wang, and J. Zhang, "VFL: A verifiable federated learning with privacy-preserving for big data in industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 18, no. 5, pp. 3316–3326, May 2022.

[29] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Proc. Annu. Int. Cryptol. Conf.*, 2001, pp. 213–229.

[30] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 1310–1321.

Yanli Ren received the M.S. degree in applied mathematics from Shaanxi Normal University, Xi'an, China, in 2005, and the Ph.D. degree in computer science and technology from Shanghai Jiao Tong University, Shanghai, China, in 2009.

She is currently a Professor with the School of Communication and Information Engineering, Shanghai University, Shanghai. She has published more than 80 quality papers, including publications in IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE TRANSACTIONS ON MULTIMEDIA, AsiaCCS, and INDOCRYPT. Her research interests include applied cryptography, secure outsourcing computation, blockchain security, AI security, and network security.

Yerong Li received the B.S. degree in electronic information from Shanghai University, Shanghai, China, in 2020, where she is currently pursuing the master's degree with the School of Communication and Information Engineering.

Her research interests are applied cryptography and privacy-preserving machine learning.

Guorui Feng received the B.S. and M.S. degrees in computational mathematic from Jilin University, Changchun, China, in 1998 and 2001, respectively, and the Ph.D. degree in electronic engineering from Shanghai Jiaotong University, Shanghai, China, in 2005.

From January 2006 to December 2006, he was an Assistant Professor with East China Normal University, Shanghai. During 2007, he was a Research Fellow with Nanyang Technological University, Singapore. He is currently with the School of Communication and Information Engineering, Shanghai University, Shanghai. His current research interests include image processing, image analysis, and computational intelligence.

Xinpeng Zhang received the B.S. degree in computational mathematics from Jilin University, Changchun, China, in 1995, and the M.S. and Ph.D. degrees in communication and information system from Shanghai University, Shanghai, China, in 2001 and 2004, respectively.

Since 2004, he has been with the faculty of the School of Communication and Information Engineering, Shanghai University, where he is currently a Professor. He was with the State University of New York at Binghamton, Binghamton, NY, USA, as a Visiting Scholar from January 2010 to January 2011, and Konstanz University, Konstanz, Germany, as an experienced Researcher sponsored by the Alexander von Humboldt Foundation from March 2011 to May 2012. His research interests include multimedia security, image processing, and digital forensics. He has published more than 200 papers in these areas.

Authorized licensed use limited to: University of Electronic Science and Tech of China. Downloaded on July 08,2023 at 09:06:36 UTC from IEEE Xplore. Restrictions apply.