Computer Science and Information Systems 22(4):1405–1432 https://doi.org/10.2298/CSIS250221057Z

VSAF: Verifiable and Secure Aggregation Scheme for Federated Learning in Edge Computing

Shiwen Zhang[1,2], Feixiang Ren[1,2], Wei Liang[1,2], Kuanching Li[1,2,⋆], and Al-Sakib Khan Pathan[3]

[1] School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China
[2] Sanya Research Institute, Hunan University of Science and Technology, Sanya 572024, China
shiwenzhang@hnu.edu.cn
rfx.point@mail.hnust.edu.cn
wliang@hnust.edu.cn
aliric@hnust.edu.cn
[3] Department of Computer Science and Engineering, United International University, Dhaka 1212, Bangladesh
spathan@ieee.org

Abstract. Federated Learning (FL) has gained attention for its promising privacy protection. In FL, clients train local gradients on their data without sharing raw data to update the global model. However, security issues persist. Attackers can infer original data from local gradients, compromising privacy, while a malicious cloud server may tamper with uploaded parameters, leading to incorrect aggregation. Considering this, we focus on the above issues in FL: (1) privacy protection of the parameters uploaded by clients and (2) verification of the correctness of the aggregated result from a cloud server. In response to these issues, this article proposes VSAF, a verifiable and secure aggregation scheme for federated learning in edge computing. Using a linear homomorphic hash function, we design a lightweight verification algorithm for aggregated gradients. To protect gradient privacy, we combine the Bloom filter and Shamir's secret sharing to design a single masking protocol. Detailed analyses and experiments demonstrate the security and efficiency of the proposed scheme.

Keywords: Federated Learning, Privacy-preserving, Correctness Verification, Edge Computing.

1. Introduction

As the Internet of Things (IoT), along with mobile devices, becomes more widespread, more and more computing tasks can be processed on the edge devices [6,33,37]. In recent years, edge devices have become increasingly intelligent and more powerful, enabling the use of edge computing [26]. This allows for transferring computing tasks and stored data from central servers to devices at the network's edge [34, 35]. As a result, both computing efficiency and data privacy protection are improved. Therefore, how to effectively use

⋆ Corresponding author

these large numbers of IoT devices and the data they generate has become a hot research topic in academia and industry [26, 36]. Many institutions and enterprises are conducting machine learning on edge nodes [5, 7, 11]. For example, the Google team has used its users' smartphones for training to predict the next word on the virtual keyboard and perform a music recognition search [11]. However, a critical and sensitive issue is that users would not like Google to access their private data for these services.

To address such an issue, Federated Learning (FL) has received widespread attention since it has performed well in privacy protection and ensured data security. FL is a technology that can achieve distributed machine learning whilst protecting data privacy [25, 27, 31]. In edge computing scenarios like vehicular networking, healthcare, and finance, FL has been widely used to achieve cross-device, cross-platform, and cross-institutional machine learning cooperation. Nonetheless, FL still has two issues that need to be addressed.

The first is how to prevent privacy leakage [12, 18, 24, 41]. Attackers or servers can infer information about the dataset used for training by the client from the gradients uploaded by the client, resulting in privacy leakage of the client. Some works depict this attack. For instance, Melis *et al.* [18] proved that the uploaded gradients may expose the privacy of clients' local data. Zhu *et al.* [41] trained on an image dataset and proved that the client's gradients would leak information about the images in its private dataset. Again, the second issue effectively verifies the aggregated result's correctness [29, 38]. It is possible for a malicious central server to modify the aggregated result of gradients and return incorrect results, leading to a failure of convergence of the training model. In addition to this, a lazy server may deliberately omit some users' gradients to save computational overhead and only aggregate the gradients of some other users, resulting in an inaccurate global model and affecting the model's convergence and efficiency [16, 29]. Hence, verifying the correctness of aggregated results while simultaneously protecting users' data privacy in an edge computing environment is challenging.

To solve the privacy protection problem, some researchers have contributed their mechanisms [1, 2, 4, 20] on privacy-protected FL. Phong *et al.* [20] used additive homomorphic encryption to protect model parameters and achieve secure aggregation. However, homomorphic encryption generates higher communication and computational overheads. M. Abadi *et al.* [1] designed a deep learning framework that integrates differential privacy and a gradient descent algorithm to protect users' data privacy. Nevertheless, differential privacy can cause an accuracy loss problem. Another more direct method is to blind the gradient directly. Keith Bonawitz *et al.* [4] introduced a double masking scheme, based on the (t, n) threshold secret sharing, to protect users' gradients. However, this scheme has a considerable restriction on the threshold t; that is, if the threshold t is less than or equal to $\lfloor \frac{n}{2} \rfloor$, the cloud server may divide all users into two sets: A and B on average, deceive users in set A (or B) to obtain the secret shares of users in set B (or A); thereby infringing on the privacy of users in set B (or A). Of course, if the value of t is too small, the cloud server can divide more sets and then resist such a privacy attack. In addition, the double masking scheme masks the user's local gradient twice to protect user privacy, and two masks will generate more computational overhead and communication overhead to a certain extent.

To verify the aggregated result's correctness in FL, existing schemes [16, 29] use homomorphic hash functions HH to verify the aggregated result. Each user uses HH to

generate proof for local gradients and uploads the hash values to the cloud server. The users receive the aggregated result of proofs from the server and verify the correctness of the gradients' aggregated result by determining if the aggregated result of proofs is consistent with the proof of the gradients' aggregated result. However, researches [16,29] allows the cloud server to collude with some users, and then the cloud server can obtain the homomorphic hash function owned by the users. Thus, after maliciously modifying the gradients' aggregated result, the server can change the aggregated result of hash values to prevent users from detecting this malicious behavior [9].

To avoid this problem, researchers [8,13,40] delegate the authority to aggregate proofs to each user, as each user sends gradient's proof to the cloud server, and the cloud server broadcasts them to the other users. Next, each user verifies the other users' proofs. After verification passes, all proofs are aggregated and used to verify the correctness of the aggregated result. These approaches increase the users' computational and communication overhead and cannot resist lazy servers' deletion attacks (Deletion attack refers to the sluggish behavior of lazy servers to save computing resources and communication overhead by only summarizing or broadcasting some users' data. From the user's perspective, it is as if the server has 'deleted' some users' data). Therefore, these approaches may cause inaccurate aggregated results and affect model convergence efficiency.

To address the above issues, we propose a verifiable and secure aggregation scheme for federated learning in edge computing called VSAF. To protect the privacy of user gradients, we design a single masking protocol based on the (t, n) threshold secret sharing mechanism and the Bloom technique. This protocol supports the dropout of some users while also protecting their privacy. To verify the correctness of the aggregated result while also discovering lazy servers, we developed a lightweight verification algorithm. This algorithm combines a homomorphic hash function with dual servers, reducing the computation and communication overheads of the user for verification. The users send the proofs for verification to the Trusted Authority (TA) and local gradients to the aggregation server, preventing the lazy and tampering behavior of the server by leveraging the mutual distrust between these two servers. To optimize verification efficiency, we plan to outsource the verification operation to the TA to reduce the user's computation overhead. In addition, since we use the homomorphic hash function for verification, the overhead for verification is independent of the gradient dimension, which can reduce the user's computation and communication loads.

The key contributions of this work are as follows:

(1) Design a single masking scheme to protect the privacy of user gradient and also tolerate the dropout of some users. Compared with the double masking scheme, we lift the restriction on the threshold t and simultaneously reduce some communication and computation overheads.
(2) Put forward a lightweight verification algorithm that leverages linear homomorphic hash function to realize the verification for the correctness of aggregated results. This method gives the aggregation authority of hash values to a trusted authority, which reduces the computation overhead of user verification, and it can also detect the lazy aggregation server in time.
(3) Achieve that the communication overhead for verification is independent of the gradient dimension, the dropout rate, and the number of users; thereby diminishing the communication overhead for verification.

(4) Implementation and evaluation VSAF. The comprehensive theoretical analysis and experimental results of the proposed scheme demonstrate its security and efficiency.

The remainder of this article is organized as follows: we first review the related work in Section 2 and introduce the preliminaries in Section 3. Then, we depict the problem statement in Section 4, followed by the description of our scheme VSAF in Section 5. After that, we analyze the security of VSAF in Section 6, and evaluate the performance in Section 7, and finally, concluding remarks and future directions are given in Section 8.

2. Related Work

2.1. Privacy Protection Schemes in Federated Learning

To address the privacy leakage problem caused by intermediate parameters in federated learning, many privacy protection schemes [1,4,15,20,23] have been proposed. To prevent attackers from recovering the training set from intermediate parameters through numerical methods, Phong *et al.* [20] use additive homomorphic encryption to protect model parameters and achieve secure aggregation. However, all participants employed the same key for the encryption and decryption of the model parameters. If any participant leaks the key pair to the attackers, the privacy of all participants will be at risk of being revealed. In addition, homomorphic encryption has a high computational overhead. Li *et al.* [15] use homomorphic encryption to encrypt the training data and directly train on the ciphertext, so thus, the data privacy is protected, but the computational overhead is still significant.

Bonawitz *et al.* [4] propose a double masking scheme, implemented based on secure multi-party computation and pseudo-random generator. This scheme can achieve privacy protection for the parameters uploaded by participants while also achieving robustness for users who are dropping out. However, it incurs high communication and computational overheads. Shokri *et al.* [23] propose a joint deep learning framework that prevents the server from directly accessing the training dataset and uses differential privacy to perturb some of the gradients. Abadi *et al.* [1] introduce a deep learning scheme that integrates differential privacy with a gradient descent algorithm, adding appropriate Laplace noise during gradient descent so that the local gradient satisfies differential privacy. However, the differential privacy can lead to the model's accuracy loss. Although these works [1,23] achieve privacy protection with smaller computational and communication overheads, it is necessary to balance privacy and accuracy.

2.2. Verifiable Aggregation Schemes in Federated Learning

Several researches on verifiable federated learning have been done in recent years, such as [10, 16, 19, 30, 32, 39, 40], and many of them are verifiable federated learning schemes focused on verifying the correctness of aggregated results [10, 16, 19, 30, 40]. Specifically, these schemes detect malicious or lazy dishonest behavior of aggregation servers by verifying the correctness of aggregated results. Other schemes [32, 39] focus on various aspects of verification, which are mainly related to detecting server failure issues and verifying the integrity of the gradients.

In [10,19,30], the server returns the aggregated result and its proof to the users, so they can utilize the proofs to verify the correctness of aggregated results and justify whether

the server is trusted or malicious. Zhou *et al.* [40] utilized homomorphic hashing combined with signature techniques to verify the aggregated result, in which all clients must take part in the verification process to verify the correctness of the parameters of other clients. However, with the increase in the number of participants, the time cost of the verification process increases. Li Lin *et al.* [16] proposed a discrete logarithm-based verification scheme that verifies the correctness of the aggregated result and discovers inert cloud servers simultaneously. However, in the above schemes, each user must validate other users' data before verifying the accuracy of the aggregated result. Furthermore, the increase in the number of users will lead to an increase in verification operations. Therefore, the larger the number of users, the higher the verification costs. At the same time, the communication overhead of the verification operation is also increased linearly with the dimension of the gradient.

To detect the server failure issues and verify the integrity of the gradients, Zhao *et al.* [39] and Zhang *et al.* [32] have proposed different schemes. Zhao *et al.* [39] introduced the PVD-FL framework that employs a cryptographic-based matrix multiplication (EVCM) algorithm for the encryption and verification of parameters. However, it can only verify the incorrect aggregated result caused by problems such as insufficient computing power and device failure of honest users and does not support detection and verification of malicious behavior by the dishonest cloud server. Zhang *et al.* [32] designed a verifiable federated learning scheme based on an online/offline signature method that realizes the integrity verification of gradients during the transmission process. However, this scheme cannot verify the incorrect aggregated result or detect the malicious behavior of the cloud server.

Unlike previous works, we propose a verifiable and secure aggregation scheme for federated learning in edge computing (VSAF). The proposed scheme can protect users' privacy, detect the aggregation server's tampering and lazy behavior, and effectively reduce verification communication and computational overheads.

3. Preliminaries

3.1. Federated Learning

The general federated learning framework is shown in Fig. 1, existing one cloud server and N clients. Each client is a user, denoted as $u_{i:1\leq i\leq N}$, who trains a local gradient $x_{i:1\leq i\leq N}$ based on the local dataset $D_{i:1\leq i\leq N}$ and sends x_i to server.

During this training process, the user u_i's gradient is typically computed using the Stochastic Gradient Descent (SGD) algorithm. Specifically, u_i first uses the global model W and the local dataset D_i to compute the gradient, x_i

$$x_i = \nabla L(W, D_i) \tag{1}$$

Here, x_i represents the direction of the steepest descent. The loss function is denoted by $L(\bullet)$, and its derivative is represented by $\nabla L(\bullet)$. The inputs to this function are the global model W and the dataset D_i.

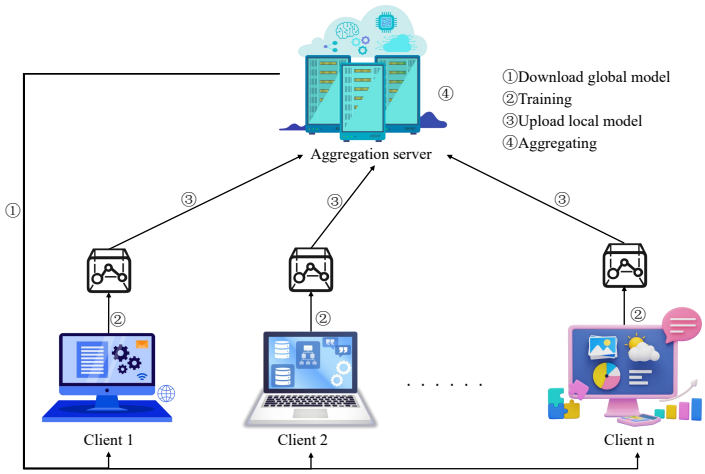

Fig. 1. The general federated learning framework

After receiving enough gradients, the cloud server acts as an aggregation server to aggregate the gradients and send users the aggregated result, which is computed as follows:

$$z = \sum_{i=1}^{N} x_i \tag{2}$$

in which z is the aggregated result.

The users update the global model W based on z and use W to carry out the next round of training. The global model update is computed as follows:

$$W = W - \eta \frac{z}{N} \tag{3}$$

in which, η is the learning rate.

Finally, repeat the above steps until the model converges or reaches the desired training accuracy.

3.2. Secret Sharing

This scheme employs Shamir's (t, N)-threshold secret sharing protocol [22]. The secret is divided into N shares, where N represents the number of users. A pre-set threshold, t, is established. The secret can only be reconstructed by gathering at least t shares. Precisely, the (t, N) threshold secret sharing protocol consists of the following steps:

(1) $\{(u_i, s_i)\}_{u_i \in U} \leftarrow S.share(s, t, U)$: This sharing algorithm divides the secret s into N shares. The inputs include the secret s, the threshold t (satisfying $t \leq \|U\|$), and the user set U ($\|U\| = N$ represents the number of users in the user set). The output is the share s_i for each user u_i.

(2) $s \leftarrow S.recon(\{(u_i, s_i)\}_{u_i \in U'}, t)$: This is a reconstruction algorithm. The inputs include the secret share s_i from users in the set U' and the threshold t, where $u_i \in U' \subseteq U$ and $t \leq u'$. The output of the algorithm is the secret s.

3.3. Homomorphic Hash

We use homomorphic hash functions to construct our verification scheme to achieve verification for the correctness of the aggregated result from the server and to defend against forgery attacks and deletion attacks from the server. Here, forgery attacks refer to when the Cloud Server (CS) forges aggregated results and sends them to the users.

If a file is divided into several file blocks, the homomorphic hash function can independently compute the hash value for each block. By aggregating these hash values, we can derive the hash value for the entire file. This scheme's homomorphic hash algorithm comprises three main components:

(1) $h_i \leftarrow HH.hash(x)$: the Hash algorithm, input, and output are a d-dimensional vector x and a hash of x. In this scheme, the d-dimensional vector x refers to the user u_i local gradient, and h_i is the gradient's hash value of the user u_i. The specific calculation process of the hash value h_i can be expressed as follows:

$$h_i = \prod_{j=1}^{d} g_j^{x_j} \ mod \, p \tag{4}$$

Here, we randomly select d distinct elements from the cyclic group G of prime order q, where g_j is the jth element. x_j denotes the jth dimensional element of vector x and p is a large prime number.

(2) $HH\_proof \leftarrow HH.agregate(h_i)$: This is the aggregation algorithm for hash values. The input is the hash value of each user, and the output is the aggregated value of all the user hashes, called HH_proof in this scheme, which is used to achieve verification for the accuracy of aggregated results from the server.

(3) $HH\_va \leftarrow HH.verify(HH\_proof, z)$: This is a verification algorithm designed to check the correctness of the aggregated result from the cloud server. The inputs are the aggregated result HH_Proof of all the user hashes and the aggregated result z of all the user local gradients. The output is HH_va, representing the evaluation of the aggregated result, and its value is either 0 or 1: the result is correct, and verification passes with value 1, and 0 if otherwise. The above process is specifically formulated as follows:

$$HH\_proof \stackrel{?}{=} HH.hash(z) \tag{5}$$

3.4. Bloom Filter

This work employs the Bloom filter [3] to efficiently determine whether a query element belongs to a given set S of n elements. Using k independent hash functions $BFH_1, BFH_2, \ldots, BFH_k$, each element in S is mapped to k positions in an m-bit vector BF, initially set to 0. Adding an element involves setting the k mapped positions to 1. To check membership of an element w, $BFH_i(w)$ $(1 \leq i \leq k)$ is used to verify if all k positions are 1. If any position is 0, $w \notin S$; otherwise, w is assumed to be in S. While Bloom filters optimize query efficiency and memory usage, they are prone to false positives, where $w \notin S$ but is falsely identified as a member. The false positive rate is $(1 - e^{\frac{nk}{m}})^k$, minimized to 2^{-k} when $k = (\ln 2) \cdot \frac{m}{n}$.

In this scheme, we use the Bloom filter to defend the server against deceiving attacks by verifying that the user requested by the server is dropped. The process unfolds as follows:

(1) $BF_i \leftarrow BFH(u_i)$: This is a hash algorithm, where the input is the identity number u_i of the online user and is mapped onto the vector BF_i through k independent hash functions $BFH_{i:1\leq i\leq k}$. BF_i represents the online status of the user u_i and is the proof that the user is online.
(2) $BF \leftarrow BF.aggregate(BF_i)$: This aggregation algorithm will map all online users to a vector BF. The input is the mapping vector for each user, and the output vector represents the online status of all users. The calculation process of vector BF is:

$$BF = \bigcup_{u_i \in \mu} BF_i \tag{6}$$

where μ denotes the set of online users.
(3) $BF\_va \leftarrow BF.verify(BF, S)$: This is a verification algorithm. The input is the set S of some users and the vector BF that reflects the user's online status. This algorithm is used to verify whether the users in set S belong to online users. The output value is 0 or 1. If $BF\_va = 0$, it indicates that users in set S are not online. If $BF\_va = 1$, it indicates that the users in set S are online users.

3.5. Key Agreement

This scheme uses the Diffie-Hellman (DH) key agreement protocol to create a secure channel between any two users, and this channel is used to negotiate the generation of shared random numbers. Specifically, we acknowledge that a group G has a prime order q, and g is the generator of G. Subsequently, the DH protocol in this scheme is composed of these two algorithms:

(1) $(sk_i, g^{sk_i}) \leftarrow DH.gen(G, p, q)$: This algorithm is a key pair generation algorithm. The output key pair (sk_i, g^{sk_i}) is the public and private keys of the user u_i respectively.
(2) $sk_{i,j} \leftarrow DH.agree(sk_i, g^{sk_j})$: The algorithm is a shared key generation algorithm. The inputs are the private key sk_i and the public key g^{sk_j}, which belong to user u_i and user u_j, respectively. The output is the shared key $s_{i,j}$, enabling secure communication between users u_i and u_j.

4. Problem Statement

In this section, we initially present the system model, followed by an introduction to the threat model, our design goals, and an overview of our VSAF.

4.1. System Model

The proposed system contains three types of entities: N users provide data at the edge, a cloud server (CS), and a trusted authority (TA), as depicted in Fig.2 the system model of the proposed VSAF.

Users: The participants who join federated learning are typically computing nodes at the network edge, such as smartphones, computers, and other IoT devices. Each user has a local dataset and is trained based on the local dataset.

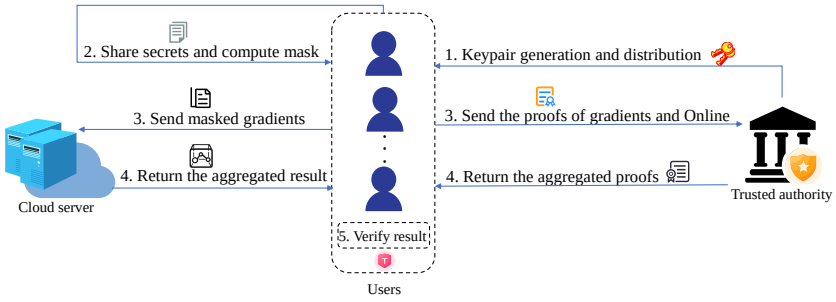

Fig. 2. System model of VSAF

Cloud Server (CS): In this system, the cloud server collects the local gradients that the users upload and aggregates the gradients. Then, the server sends the aggregated result to the users. Using this result, users update the global model and proceed to the next round of training.

Trusted Authority (TA): TA mainly generates and distributes initialization parameters in this system. Specifically, TA generates some initialization parameters for the user, such as the initial global model and the key pairs for encryption. In addition, it is responsible for generating the users' online proofs, which can help the users resist the deceiving attack of the server. TA also generates the aggregation verification proof, assisting the users to verify the correctness of the aggregated result from the server.

4.2. Threat Model

We assume the following threat model:

(1) **Malicious and lazy CS:** We consider that CS may tamper with users' data or omit some users' data to save computational load during the aggregation process. Therefore the CS in this scheme is not honest but a curious server.
(2) **Semi-honest Users:** In this system, the users perform the protocol honestly despite interested in other users' data, and attempting to get access to other users' data.
(3) **TA:** TA is trustworthy, and will strictly adhere to the protocol, and will strive to maintain the privacy of users.

In this scheme, we have the following assumptions for the threat model: the CS cannot collude with the TA but may collude with fewer than t users and tamper with or delete user-uploaded data during aggregation. The CS can also launch deception and deletion attacks during parameter aggregation and verification to extract users' local gradients or compromise the aggregated result.

Deceiving attacks: While resolving the dropout problem, since the CS is not trustworthy, it may deceive the online users by falsely claiming that certain users have dropped out to obtain their secret shares. Then, CS reconstructs the masks of these users, enabling it to compute their local gradients and thus violate their privacy.

Deletion attacks: In typical verification schemes, users send their local gradients and corresponding proofs to the CS, which then aggregates and distributes the results along

with the proofs to the users for verification. However, to reduce computational and communication overhead, the lazy CS may aggregate only the gradients from a subset of users and send the aggregated result along with their proofs. It appears the server has "deleted" the data from specific users, and users cannot detect this deletion through the verification algorithm.

4.3. Design Goals

Specifically, this work should meet the following design goals:

- **Privacy.** Our proposed VSAF aims to protect users' gradients' privacy, preventing adversaries from recovering users' sensitive data from the gradients and defending against deceiving attacks.
- **Verification.** VSAF should be able to detect deletion attacks to ensure the correctness of the aggregated results.
- **Efficiency.** VSAF aims to achieve efficient communication and computation for verification, saving on communication and computational overheads.

4.4. Overview of VSAF

The processes of VSAF are divided into four rounds: negotiation and distribution of keys, generation and distribution of shares, generation and uploading of ciphertexts and proofs, and aggregation and verification of parameters. We will explain these four rounds in detail in the next section.

(1) **Negotiation and Distribution of Keys (Round 1):** TA generates key pairs for each user and public parameters HH_{pp} and BF_{pp} for the homomorphic hash function and bloom filter algorithm. Then, TA distributes key pairs and public parameters to the corresponding users.
(2) **Generation and Distribution of Shares (Round 2):** All users share their one private key through Shamir's secret sharing technique. Then, the users encrypt the shares and distribute the ciphertext of the shares to the corresponding users with the help of the server.
(3) **Generation and Uploading of Ciphertexts and Proofs (Round 3):** According to the DH protocol, any two users negotiate a shared key and then use this key as a seed to generate a mask for encrypting the local gradient. Each user uses the bloom filter and homomorphic hash function to generate online proof and verification proof. Finally, the ciphertexts and proofs are sent to CS and TA.
(4) **Aggregation and Verification of Parameters (Round 4):** The users verify the CS's request based on the online proof and upload the secret shares of the dropped users. CS recovers the masks using the secret shares, then derives the unbiased aggregated result based on these masks, and finally sends this result to online users. The users verify the correctness of the aggregated result based on the proof of verification.

Implementation of VSAF

Round 1 (*Negotiation and Distribution of Keys*)

TA :

- Generate two key pairs $(N_i^{pk}, N_i^{sk}) \leftarrow DH.gen(G,p,q)$ and $(M_i^{pk}, M_i^{sk}) \leftarrow DH.gen(G,p,q)$ for each user u_i.
- Generate public parameter HH_{pp} of the linear homomorphic hash algorithm.
- Generate public parameter BF_{pp} of the bloom filter algorithm.
- Send (N_i^{pk}, N_i^{sk}), (M_i^{pk}, M_i^{sk}), HH_{pp} and BF_{pp} to the corresponding users.

$User\ u_i$:

- Receive $\{(N_i^{pk}, N_i^{sk}), (M_i^{pk}, M_i^{sk}), u_i^{id}, HH_{pp}, BF_{pp}\}$ from TA.
- Send two public keys (N_i^{pk}, M_i^{pk}) to CS, and use the CS to broadcast the public keys (N_i^{pk}, M_i^{pk}) to other users.

CS :

- Receive (N_i^{pk}, N_i^{sk}) from users. Set μ_1 as the set of users whose messages are received by CS, in which $\mu_1 \subseteq \mu$ and μ_1 is the set of all users. Moreover, ensure that $|\mu| \geq t$.
- Generate the id number u_i for user $u_i \in \mu_1$, so that CS can number the users who sent the messages, where all the u_i are different from each other and used to generate users' online proofs.
- Send $\{u_i, N_i^{pk}, M_i^{pk}\}_{u_i \in \mu_1}$ to each user $\in \mu_1$.

Round 2 (*Generation and Distribution of Shares*)

$User\ u_i$:

- Receive $\{u_i, N_i^{pk}, M_i^{pk}\}_{u_i \in \mu_1}$ from CS.
- Generate the shares of N_i^{sk} as $\{(u_i, s_{i,j})\}_{u_j \in \mu_1} \leftarrow S.share(N_i^{sk}, t, |\ \mu_1\ |)$, where $s_{i,j}$ is the share of user u_i to user u_j.
- Compute $C_{i,j} \leftarrow DH.Enc(DH.agree(M_i^{sk}, M_j^{pk}), u_i\ ||\ u_j\ ||\ s_{i,j})$, where $C_{i,j}$ is the ciphertext of user u_i to user u_j.
- Send the ciphertext $\{C_{i,j}\}_{u_j \in \mu_1}$ to CS.

CS :

- Receive the ciphertext $\{C_{i,j}\}_{u_j \in \mu_1}$ from user $u_i \in \mu_2$, in which $\mu_2 \subseteq \mu_1$ and μ_2 is the set of users whose messages are received by CS. In addition, ensure that $|\ \mu_2\ | \geq t$.
- Broadcast the ciphertext $\{C_{i,j}\}_{u_i \in \mu_1}$ to user $u_j \in \mu_2$.

Round 3 (*Generation and Uploading of Ciphertexts and Proofs*)

$User\ u_i$:

- Receive the ciphertext $\{C_{j,i}\}_{u_i \in \mu_2, u_j \in \mu_1}$ and ensure that $\mu_2 \subseteq \mu_1$ and $|\ \mu_2\ | \geq t$.
- Negotiate the shared key $S_{i,j} \leftarrow DH.agree(N_i^{sk}, N_j^{pk})$ with user $u_j \in \mu_2$ according to the DH protocol, where $S_{i,j}$ is the shared key between user u_i and user u_j.
- Compute the masks $msk_{i,j} \leftarrow PRG(S_{i,j})$, where $PRG()$ is a pseudorandom generator.
- Get a local gradient x_i, after training on local dataset.
- Mask the gradient x_i as $\widehat{x}_i \leftarrow x_i + \sum_{u_j \in \mu_2 : i<j} msk_{i,j} - \sum_{u_j \in \mu_2 : i>j} msk_{i,j}$.
- Compute the verification proof of the local gradient $h_i \leftarrow HH(x_i)$.
- Compute the online proof $BF_i \leftarrow BFH(u_i^{id})$, where u_i^{id} is the identity number for user u_i.
- Send the encrypted gradient $\{\widehat{x}_i\}$, and proofs $\{h_i, BF_i\}$ to CS and TA, respectively.

CS :

- Receive $\{\widehat{x}_i\}$ from user $u_i \in \mu_3$, where μ_3 is the set of users who send messages to CS and TA. Besides, ensure that $\mu_3 \subseteq \mu_2$, $|\ \mu_3\ | \geq t$.
- Send a list of $\mu_2 \backslash \mu_3$ to each user in μ_3.

TA :

- Receive $\{h_i, BF_i\}$ from user $u_i \in \mu_3$, where μ_3 is the set of users who send messages to CS and TA. Furthermore, ensure that $\mu_3 \subseteq \mu_2$, $|\ \mu_3\ | \geq t$.
- Compute the verification proof of the aggregated result as $HH\_proof \leftarrow HH.aggregate(h_i)$.
- Compute the online proof of all the users in the set μ_3 as $BF \leftarrow BFH.aggregate(BF_i)$.
- Send the proofs $\{HH\_proof, BF\}$ to each user $u_i \in \mu_3$.

Round 4 (*Aggregation and Verification of Parameters*)

$User\ u_i$:

- Receive a list of $\mu_2 \backslash \mu_3$ form CS and the proofs $\{HH\_proof, BF\}$ from TA.
- Compute $BF\_va \leftarrow BF.verify(BL, \mu_2 \backslash \mu_3)$, where BF_va is used to determine whether the users in $\mu_2 \backslash \mu_3$ are online or dropout. If the verification passes, then continue. Otherwise, abort and start over.
- Decrypt $C_{i,j}$ from dropped users as $\{u_i\ ||\ u_j\ ||\ s_{i,j}\} \leftarrow DH.Dec(DH.agree(M_i^{sk}, M_j^{pk}), C_{i,j})$.
- Send $\{u_i\ ||\ u_j\ ||\ s_{i,j}\}_{u_j \in \mu_2 \backslash \mu_3}$ to CS.

CS :

- Receive $\{s_{i,j}\}_{u_j \in \mu_2 \backslash \mu_3}$ form users $u_i \in \mu_4$, where $\mu_4 \subseteq \mu_3$ and $|\ \mu_4\ | \geq t$, otherwise, abort and start over.
- Reconstruct private keys $N_i^{sk} \leftarrow S.recon(\{u_i,\ s_{i,j}\}_{u_j \in \mu_4}, t)$ for users $u_i \in \mu_2 \backslash \mu_3$.
- Compute the shared key $S_{i,j} \leftarrow DH.agree(N_i^{sk}, N_j^{pk})$ and the masks $msk_{i,j} \leftarrow PRG(S_{i,j})$, where $u_i \in \mu_2 \backslash \mu_3$ and $u_j \in \mu_3$.
- Compute the aggregated result as

$$\sum_{u_i \in \mu_3} x_i \leftarrow \sum_{u_i \in \mu_3} \widehat{x}_i - \sum_{\substack{u_i\ \in\ \mu_3, \\ u_j\ \in\ \mu_2 \backslash \mu_3\ :\ i\ <\ j}} msk_{i,j} + \sum_{\substack{u_i\ \in\ \mu_3, \\ u_j\ \in\ \mu_2 \backslash \mu_3\ :\ i\ >\ j}} msk_{i,j}$$

- Broadcast the aggregated result $z = \sum_{u_i \in \mu_3} x_i$ to the user $u_i \in \mu_4$.

$User\ u_i$:

- Receive the aggregated result z form CS and verify the correctness of the aggregated result z as $HH\_va \leftarrow HH.verify(HH\_proof, z)$.
- Accept z and move to Round 1, if the $HH\_va = 1$. Otherwise, abort and start over.

Fig. 3. The detailed description of VSAF

5. The Proposed VSAF

The proposed scheme VSAF includes three types of entities: N users at the edge, a cloud server (CS), and a trusted authority (TA). In addition, it also includes four rounds: **Negotiation and Distribution of Keys (Round 1)**, **Generation and Distribution of Shares**

(Round 2), **Generation and Uploading of Ciphertexts and Proofs (Round 3)**, **Aggregation and Verification of Parameters (Round 4)**. The interaction details of each entity in different rounds are shown in Fig. 3.

5.1. Negotiation and Distribution of Keys (Round 1)

TA mainly generates key pairs (N_i^{pk}, N_i^{sk}) and (M_i^{pk}, M_i^{sk}) for users, as well as the necessary public parameters HH_{pp} and BF_{pp} for the homomorphic hash function and the bloom filter algorithm. Then, the TA assigns an identifier to each user. After that, it sends the key pairs and the public parameters to the corresponding users. Specifically, the key pairs generated by TA for user u_i are as follows:

$$(N_i^{pk}, N_i^{sk}) \leftarrow DH.gen(G, p, q) \tag{7}$$

Similarly, (M_i^{pk}, M_i^{sk}) is also generated in this way. N_i^{pk} and M_i^{pk} are public keys, and N_i^{sk} and M_i^{sk} are private keys.

After receiving messages from TA, all users will use the CS to broadcast the public keys N_i^{pk} and M_i^{pk} to other users. Set μ_1 as the users who send messages to the CS. After receiving the messages, the CS generates the identifier u_i to number the users who sent these messages and then sends $\{u_i, N_i^{pk}, M_i^{pk}\}_{u_i \in \mu_1}$ to each user. Note that all users have only two states: online or dropped.

5.2. Generation and Distribution of Shares (Round 2)

Let μ_2 be the set of users who perform secret sharing in Round 2. After completing the Round 1, the user $u_i (u_i \in \mu_2)$ secretly shares private key N_i^{sk} to other users. The purpose of secret sharing is to allow the CS to request the secret shares of the dropped users from the online users, thereby recovering the dropped users' private keys. Subsequently, the CS calculates the masks to correct its aggregated result. The secret share procedure is shown below:

$$\{(u_i, s_{i,j})\}_{u_j \in \mu_1} \leftarrow S.share(N_i^{sk}, t, \mid \mu_1 \mid) \tag{8}$$

where $S.share(\bullet)$ is the Shamir's (t, N) threshold secret sharing algorithm. In the input, N_i^{sk} is the private key of user u_i, $|\mu_1|$ is the number of shares into which N_i^{sk} needs to be divided, and *t* represents the minimum number of shares required to reconstruct N_i^{sk}. The output includes $|\mu_1|$ secret shares $s_{i,j}$ that user u_i sends to user u_j.

User u_i sends shares to the CS, which forwards them to the corresponding users. The message sent is shown below:

$$C_{i,j} \leftarrow DH.Enc(DH.agree(M_i^{sk}, M_j^{pk}), u_i \mid\mid u_j \mid\mid s_{i,j}) \tag{9}$$

where $DH.Enc(\bullet)$ is an encryption algorithm based on the DH protocol that encrypts the user identity numbers u_i, u_j, and the secret share $s_{i,j}$ using $DH.agree(M_i^{sk}, M_j^{pk})$ as the encryption key. $C_{i,j}$ denotes the ciphertext sent from user u_i to user u_j.

5.3. Generation and Uploading of Ciphertexts and Proofs (Round 3)

After Round 2, the users receive $C_{i,j}$ from CS and will decrypt them in Round 4.

After that, any two users negotiate the shared key $S_{i,j}$ between them according to the DH protocol. The negotiation procedure is as follows:

$$S_{i,j} \leftarrow DH.agree(N_i^{sk}, N_j^{pk}) \tag{10}$$

Here, $DH.agree(\bullet)$ is the shared key negotiation algorithm, which takes as input the private key N_i^{sk} of user u_i and the public key N_j^{pk} of user u_j. The output is the shared key $S_{i,j}$ between user u_i and user u_j. The shared key $S_{i,j}$ serves as the seed for a pseudo-random generator responsible for producing masks. The generation formula is as follows:

$$msk_{i,j} \leftarrow PRG(S_{i,j}) \tag{11}$$

where $PRG(\bullet)$ is a pseudo-random generator. It takes as input a shared key, which is negotiated between users u_i and u_j following the DH protocol. The output is the mask $msk_{i,j}$, which is used by user u_i and user u_j for masking the local gradient; moreover, the mask is equal in length with the gradient.

Every user performs training on their local dataset and obtains their local gradient, denoted as x_i.

After local training, user u_i encrypts the local gradient x_i as follows:

$$\widehat{x}_i \leftarrow x_i + \sum_{u_j \in \mu_2 : i<j} msk_{i,j} - \sum_{u_j \in \mu_2 : i>j} msk_{i,j} \tag{12}$$

In the abovementioned formula, $msk_{i,j}$ is the mask shared by users u_i and u_j, where both u_i and u_j belong to μ_2. u_i uses $msk_{i,j}$ to mask the local gradient x_i, and then obtains the encrypted gradient x_i by adding $msk_{i,j}$ to the local gradient x_i (where i<j), subtracting $msk_{i,j}$ (where i>j).

Next, the user u_i calculates both the local gradient's verification proof and the online proof:

$$h_i \leftarrow HH(x_i) \tag{13}$$

$$BF_i \leftarrow BFH(u_i) \tag{14}$$

Finally, u_i sends the encrypted gradient $\{\widehat{x}_i\}$ to the CS, the verification proof, and the online proof $\{h_i, BF_i\}$ to the TA.

Let μ_3 be the set of users who send messages to CS and TA. The CS sends a list of $\mu_2 \backslash \mu_3$ to each user in μ_3, requesting the offline users' shares for unmasking the aggregated result. Here, $\mu_2 \backslash \mu_3$ represents the users in set μ_3 but not in set μ_2.

TA aggregates the verification proofs h_i and online proofs BF_i of all users and broadcasts the two aggregated proofs to users. The aggregation mechanism is as follows:

$$HH\_proof \leftarrow HH.aggregate(h_i) \tag{15}$$

$$BF \leftarrow BFH.aggregate(BF_i) \tag{16}$$

where HH_proof denotes the verification proof of the aggregated result, which is used to verify the correctness of the gradient aggregated by CS. BF denotes the online proof of all users in the set μ_3, and any user can verify whether a particular user is online through BF.

5.4. Aggregation and Verification of Parameters (Round 4)

After receiving the list $\mu_2 \backslash \mu_3$ from the CS, to defend against the CS's deceiving attack, the users will verify whether the users in $\mu_2 \backslash \mu_3$ are dropped users. The verification procedure is as follows:

$$BF\_va \leftarrow BF.verify(BL, \mu_2 \backslash \mu_3) \tag{17}$$

The $BL.verify(\bullet)$ algorithm is a verification algorithm, used to verify whether the users in $\mu_2 \backslash \mu_3$ are dropped out. The inputs are the set of dropped users $\mu_2 \backslash \mu_3$ sent by CS and the users' online proof BL sent by TA. If the output is 1, it indicates that all users in the set $\mu_2 \backslash \mu_3$ are offline, and the verification is successful. If not, the verification fails, and users decline to send the shares of users in the set $\mu_2 \backslash \mu_3$ to the CS.

In Round 3, each online user receives the ciphertext $C_{i,j}$ of the secret shares from other users. Therefore, after the verification is passed, users will only decrypt the $C_{i,j}$ of those who have been dropped, specifically those who have shared their secret shares but have not uploaded the ciphertext of their gradients. The decryption algorithm is expressed as follows:

$$\{u_i \mid\mid u_j \mid\mid s_{i,j}\} \leftarrow DH.Dec(DH.agree(M_i^{sk}, M_j^{pk}), C_{i,j}) \tag{18}$$

where $DH.Dec(\bullet)$ is a decryption algorithm based on the DH protocol that decrypts the ciphertext $C_{i,j}$ using $DH.agree(M_i^{sk}, M_j^{pk})$ as the decryption key.

Next, users send the shares of the dropped users to CS. Let μ_4 represent the set of users who send information to the CS. CS receives shares from at least t users; otherwise, it stops. After receiving enough shares, CS reconstructs the private key N_i^{sk} of the dropped user u_i, where $u_i \in \mu_2 \backslash \mu_3$. The reconstruction algorithm is as follows:

$$N_i^{sk} \leftarrow S.recon(\{u_i,\ s_{i,j}\}_{u_j \in \mu_4}, t) \tag{19}$$

CS calculates the masks of the dropped users based on N_i^{sk}, and calculated as follows:

$$msk_{i,j} \leftarrow PRG(DH.agree(N_i^{sk}, N_j^{pk})) \tag{20}$$

For the subscripts i and j, where $u_i \in \mu_2 \backslash \mu_3$, and $u_j \in \mu_3$
Afterwards, CS uses $msk_{i,j}$ to correct the aggregated result:

$$\sum_{u_i \in \mu_3} x_i \leftarrow \sum_{u_i \in \mu_3} \widehat{x_i} \ - \sum_{\substack{u_i \in \mu_3, \\ u_j \in \mu_2 \backslash \mu_3 : i < j}} msk_{i,j} \ + \sum_{\substack{u_i \in \mu_3, \\ u_j \in \mu_2 \backslash \mu_3 : i > j}} msk_{i,j} \tag{21}$$

Lastly, CS sends the aggregated result $z = \sum_{u_i \in \mu_3} x_i$ to each user. After receiving z, the users will verify it. The verification process is as follows:

$$HH\_va \leftarrow HH.verify(HH\_proof, z) \tag{22}$$

where $HH.verify(\bullet)$ represents the aggregated result verification algorithm, and the inputs are the verification proof HH_proof and the aggregated result z. If the output is 1, it indicates the aggregated result is accurate, the verification is successful, and users accept the aggregated result z, moving on to the next training round. If not, the verification fails, leading to a halt in training and a restart.

6. Security Analysis

This section will analyze and prove the security of our scheme. Firstly, we will demonstrate that our scheme will protect user local gradients' privacy (Input Privacy). Secondly, we will conduct a security analysis on server forgery and deletion attacks. Finally, we will demonstrate that our verification scheme is correct.

6.1. Privacy Protection of the Gradients

Firstly, as can be inferred from the previous text, we employ a single masking scheme to ensure the privacy and security of the user's local gradient. Every user masks the local gradient as:

$$\widehat{x_i} = x_i + \sum_{u_j \in \mu_2, i<j} msk_{i,j} - \sum_{u_j \in \mu_2, i>j} msk_{i,j} \tag{23}$$

There is a lemma that, if we have some uniformly random numbers added to the inputs of users, the result will appear uniformly random.

In our threat model, the server is honest but curious. Moreover, it may collude with fewer than $t-1$ users to infer a specific user's input privacy. In addition, if the server is malicious, the subsequent sections on correction of the aggregated result and correction of verification will ensure that our scheme is secure. Next, before proving our scheme's input privacy, we must introduce some useful notation. We denote Cloud Server by the set S, the n users participating in federated learning by the set U, and introduce a security parameter k for the cryptographic primitives, using t to denote the threshold in Shamir's secret sharing. Because of the user dropout problem, we denote by U_i the set of users whose local parameters are received by the CS in round $i-1$. The number of users may change each round as users can drop out of training anytime. Hence, we have $U_3 \subseteq U_2 \subseteq U_1 \subseteq U$. We denote by $U_i \backslash U_{i+1}$ the users whose messages were received by the server in round $i-1$, but not received in round i.

Let $W \subseteq U \cup S$ be a set of corrupt parties. The combined perspective of all parties within W is characterized by a random variable $REAL_W^{U,t,k}(x_U, U_1, U_2, U_3)$, where k stands for a security parameter and t is the threshold in our protocol. This view includes the parties' input in W, randomness, and all communications received from parties outside of W. Additionally, the party will remain receiving messages until it drops out and stops receiving messages. Then, we will postulate two theorems to discuss the security of the input privacy in our protocol. In these theorems, one considers the collusion of active adversarial users, and the other is based on the collusion between Cloud Servers and users. Ultimately, they can show that any collusion between these parties cannot infringe on the privacy of others.

Theorem 1: (Safeguarding against collaborative assaults from multiple users) For all k, t, x_U and $U_3 \subseteq U_2 \subseteq U_1 \subseteq U$, a PPT simulator SIM exists with an output indistinguishable from $REAL_W^{U,t,k}$:

$$REAL_W^{U,t,k}(x_U, U_1, U_2, U_3) \equiv SIM_W^{U,t,k}(x_U, U_1, U_2, U_3) \tag{24}$$

Proof: As we only consider the collusion between multiple users excluding the Cloud Server, the joint view of the parties in set W is independent of the inputs from users not

in W. A perfect simulation can be achieved by having the simulator operate all honest users on false inputs while running the honest but curious users on their actual inputs. As the messages received by users from Cloud Server only include the set of the online users and final aggregation but do not (contain) the true value of $\widehat{x_n}$, the simulator can utilize random numbers to mask all honest users' inputs, instead of using true values. Hence, the parties in W will be unable to determine if the input from honest users is true or dummy. Ultimately, the simulated perspective of parties in set W is identical to the actual perspective $REAL_W^{U,t,k}$.

Theorem 2: (Guarding against collaborative assaults from users and the Cloud Server) For all $k, t, U, x_U, W \subseteq U \cup S, n_W = |W \backslash S|, n_W < t$, and $U_3 \subseteq U_2 \subseteq U_1 \subseteq U$, a PPT simulator SIM exists, generating an output that, computationally, cannot be differentiated from the output of $REAL_W^{U,t,k}$:

$$REAL_W^{U,t,k}(x_U, U_1, U_2, U_3) \equiv SIM_W^{U,t,k}(x_U, U_1, U_2, U_3) \tag{25}$$

Proof: We will employ a conventional hybrid argument to provide proof for the above theorem. The approach is gradually executing an array of secure alterations on the actual view, which eventually results in the output of the simulated view being computationally identical to the output of the real view.

Hyb1: In this hybrid, regarding the interaction among users in Round 1, we employ a random numeral to substitute the shared key among any interacting entities for the message's encryption/decryption. Specifically, assuming that we fix any two users $u_i, u_j \in U_2 \backslash W, u_i \neq u_j$, u_i and u_j are honest users, so then the simulator modifies the conduct of all upright participants by employing a uniformly random numeral $r_{i,j}$ as a replacement for the shared key $DH.agree(N_i^{sk}, N_j^{pk})$ between u_i and u_j. Subsequently, u_i and u_j will use a random number $r_{i,j}$ to encrypt and decrypt the messages based on the symmetric encryption system. Eventually, the DDH assumption will ensure this hybrid is computationally identical to the real protocol.

Hyb2: In this hybrid, the simulator will use a random number (this random number is the same length as the data that honest users need to encrypt) to replace the data that all honest users $u_i (u_i \in U_2 \backslash W)$ want to encrypt. In the subsequent Rounds, when the CS requires users to upload the offline users' shares to unmask the ciphertexts of the offline users, all honest users will upload the real shares (*i.e.*, the shares of random numbers used by this hybrid). By altering the data to be encrypted, we ensure, through the properties of symmetric authenticated encryption, that this hybrid is distinguishable from the actual protocol.

Hyb3: Here, the simulator will use shares of random numbers with appropriate length instead of N_i^{SK}'s shares from all honest users u_i who are in the set U_2 but not in W. Hence, the security of Shamir's secret sharing ensures the indistinguishability of this hybrid from the actual protocol.

Hyb4: In this hybrid, we first select any two users u_i and $u_{j'}$, who are from the set $U_2 \backslash W$ and $u_i \neq u_{j'}$. Then, for the shared key $S_{i,j'} = DH.agree(N_i^{sk}, N_{j'}^{pk})$ between u_i and $u_{j'}$, the simulator selects a random number $S_{i,j'}^{'}$ uniformly for replacement. Specifically, for the user u_i, instead of sending $\widehat{x}_i = x_i + \sum_{u_j \in U_2 : i<j} PRG(S_{i,j}) - \sum_{u_j \in U_2 : i>j} PRG(S_{i,j})$, SIM submits

$\widehat{w}_i = w_i + \sum_{u_j \in U_2 \setminus \{u_{j'}\}: i<j} PRG(S_{i,j}) - \sum_{u_j \in U_2 \setminus \{u_{j'}\}: i>j} PRG(S_{i,j}) + \triangle_{i,j'} PRG(S'_{i,j})$,

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

For $u_{j'}$, there exists $\widehat{x}_{j'} = x_{j'} + \sum_{u_i \in U_2} \triangle_{i,j'} PRG(S'_{i,j})$

Subsequently, the DDH assumption certifies this hybrid as indistinguishable from the authentic protocol.

Hyb5: Based on the previous hybrid, the simulator, in this hybrid, replaces the output of $PRG(s'_{i,j'})$ with a random number that is uniformly selected. The simulator merely replaces the output of PRG, thus the security of the pseudo-random generator makes this hybrid indistinguishable from the actual protocol.

As can be seen from the previous hybrids, the distribution of these hybrids cannot be differentiated from the actual protocol, so thus, this completes and finalizes the proof.

6.2. Correctness of the Aggregated Result

By following our scheme honestly, the server can ensure that an accurate aggregated result z is ultimately obtained by the users. If the users and Cloud Server (CS) follow our scheme honestly, users can ultimately get a correct aggregated result z.

In Round 4, the CS requests the shares of dropout users after receiving parameters uploaded by the online users. After the request from the CS, is verified by the users, the CS receives the secret shares from the users who have dropped out. The CS retrieves the dropout users' private key $N_i sk$ and calculates its mask $msk_{i,j}$. Finally, the CS calculates the aggregated result:

$$z = \sum_{u_i \in U_3} x_i = \sum_{u_i \in U_3} \widehat{x}_i - \sum_{u_i \in U_3, u_j \in U_2 \setminus U_3} \triangle_{i,j} msk_{i,j} \tag{26}$$

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

Therefore, according to the above formula, CS can ultimately obtain the correct aggregated result for online users.

6.3. Correctness of Verification

Up to $t-1$ users are allowed to collude with the server in our scheme, which means that the CS can obtain the homomorphic hash function HH and users' verification scheme. Therefore, if we make the CS aggregate the hash values of the users' gradients or make the CS broadcast the hash values to each user to aggregate the hash values, a malicious CS may tamper with the aggregated result of the users' gradients and modify the aggregated result of the hash values at the same time, or a lazy CS may "delete" the local gradients and the hash values of some users in order to save computational resources, which may result in inaccurate aggregated result.

Under our assumptions above, neither of these two malicious behaviors of the CS will be detected by users, which will violate the original intention of verifiable federated learning. Therefore, our scheme delegates the operation of merging the hash values of the local

gradient updates from the users to the TA. In this scenario, the CS may tamper with the users' gradients or omit some users' gradients during aggregation to save computational overhead. Alternatively, the CS may add noise to affect the aggregated result's accuracy. However, since the TA and CS are not colluding, these actions will cause the hash value of the aggregated result to differ from the aggregated result of hash values. By comparing these two values, each user can detect these malicious behaviors by the server. Therefore, this scheme can resist malicious attacks such as forgery and deletion from CS.

7. Performance Analysis and Evaluation

7.1. Performance Analysis

In this subsection, we choose five MPC (Multi-Party Computation) based schemes, which are VerifyNet [29], VerSA [9], PFLM [13], VERIFL [8] and PVFL [40] as well as a HE (Homomorphic Encryption) based scheme (VPFL [32]) to analyze and compare the communication and computation overheads. In Table 1, n is used to represent the total number of users, d stands for the gradient dimension, and ϕ signifies the count of users who have dropped out.

Communication Overhead Analysis In Table 1, we analyze the outgoing communication of the user and the server in these schemes. [29] first proposed a secure and verifiable federated learning scheme, but it did not achieve the independence of the communication overhead for verification from the gradient dimension. Analysis of this scheme reveals that the communication overheads for each user and the server are $O(n + d)$ and $O(n^2 + nd + n + d)$, respectively. In [9], the main communication overhead lies in the user uploading the encrypted gradient and n secret shares and the server forwarding the data uploaded by the user. Thus, the communication overhead for each user is $O(n + d)$, while for the server, it is $O(n^2 + nd)$. [13] adopts a double masking protocol to protect privacy, and the user needs to receive $n - 1$ masked messages from the server to verify the correctness of the aggregated result. Thus, the communication overheads for each user and the server are $O(nd)$ and $O(n^2 + nd)$, respectively.

Again, in the work [8], all users need to receive the commitments and hash values of other online users from the server, which leads to some communication overhead, we can derive that the communication overhead for each user is $O(n + d)$, while for the server is $O(n^2 + nd)$. In [40], the communication overhead is mainly caused by the differential privacy mechanism, and the user needs to negotiate noise with other users. Therefore, its communication overhead is $O(nd)$ for each user and is $O(n^2 d)$ for the server. [32] proposes a distributed encryption of gradients algorithm, which can reduce the computational overhead of encryption but increases the communication overhead. Therefore, the communication overheads for each user and the server are $O(nd)$ and $O(n^2 d)$, respectively.

Computation Overhead Analysis. Since VPFL [32] realizes the integrity verification of gradients during the transmission process, but it cannot verify the incorrect aggregated result or detect the malicious behavior of the cloud server, we only analyze and compare the computational overhead of these schemes [8,9,13,29,40]. The comparison of the computation overheads between these schemes is shown in Table 1. The privacy-preserving

Table 1. computation and outgoing communication overhead

	Outgoing Communication		Computation Overhead	
Schemes	Each user	Server	Each user	Server
VerifyNet [29]	$O(n+d)$	$O(n^2+nd+n+d)$	$O(nd+n)$	$O(n^2+nd+d)$
VerSA [9]	$O(n+d)$	$O(n^2+nd)$	$O(nd+n)$	$O(n^2+nd)$
PFLM [13]	$O(nd)$	$O(n^2+nd)$	$O(n^2+nd+n)$	$O(n^2d)$
VERIFL [8]	$O(n+d)$	$O(n^2+nd)$	$O(nd+n+d)$	$O(n^2+nd)$
PVFL [40]	$O(nd)$	$O(n^2d)$	$O(nd+n+d)$	$O(nd)$
VPFL [32]	$O(nd)$	$O(n^2d)$	\	\
Our scheme	$O(n+d)$	$O(n^2)$	$O(nd)$	$O(n^2)$

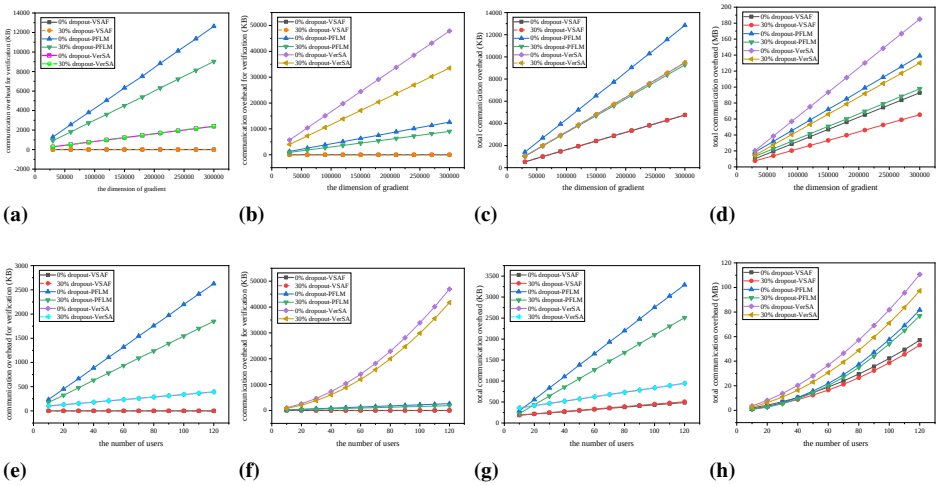

Fig. 4. The comparison of our scheme VSAF and PFLM [13] and VerSA [9] with respect to communication overhead for verification and total communication overhead. (a), (b), (c), and (d) compare the three schemes under 0% and 30% dropout rates, showing how the communication overhead varies with gradient dimension. Specifically, (a) illustrates the communication overhead for verification for each user per iteration, (b) illustrates the communication overhead for verification of CS per iteration, (c) illustrates the total communication overhead for each user per iteration, and (d) illustrates the total communication overhead of CS per iteration. (e), (f), (g), (h) compare the three schemes under 0% and 30% dropout rates, showing how the communication overhead varies with the number of users. Specifically, (e) illustrates the communication overhead for verification for each user per iteration, (f) illustrates the communication overhead for verification of CS per iteration, (g) illustrates the total communication overhead for each user per iteration, and (h) illustrates the total communication overhead of CS per iteration

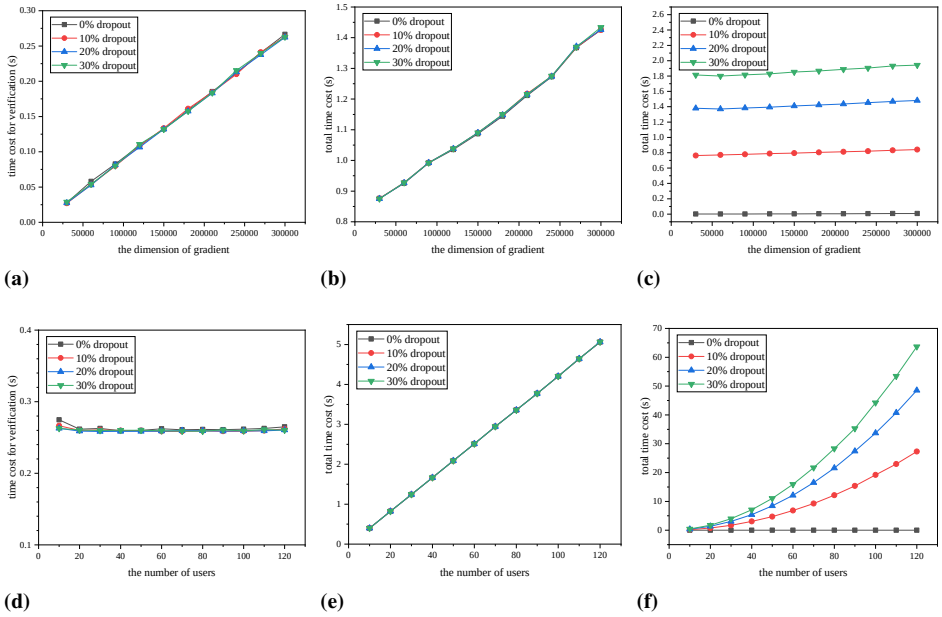

Fig. 5. Comparison of the time costs for each user and server under different dropout rates in our scheme. (a), (b), (c) show the time cost variations with gradient dimension for each user and CS in our scheme under 0%, 10%, 20%, and 30% dropout rates. Specifically, (a) illustrates the time cost for verification of each user per iteration, (b) illustrates the total time cost for each user per iteration, and (c) illustrates the total time cost for CS at each iteration. (d), (e), (f) show the time cost variations with the number of users for each user and CS in our scheme under 0%, 10%, 20%, and 30% dropout rates. Specifically, (d) illustrates the time cost for verification of each user per iteration, (e) illustrates the total time cost for each user per iteration, and (f) illustrates the total time cost for CS at each iteration

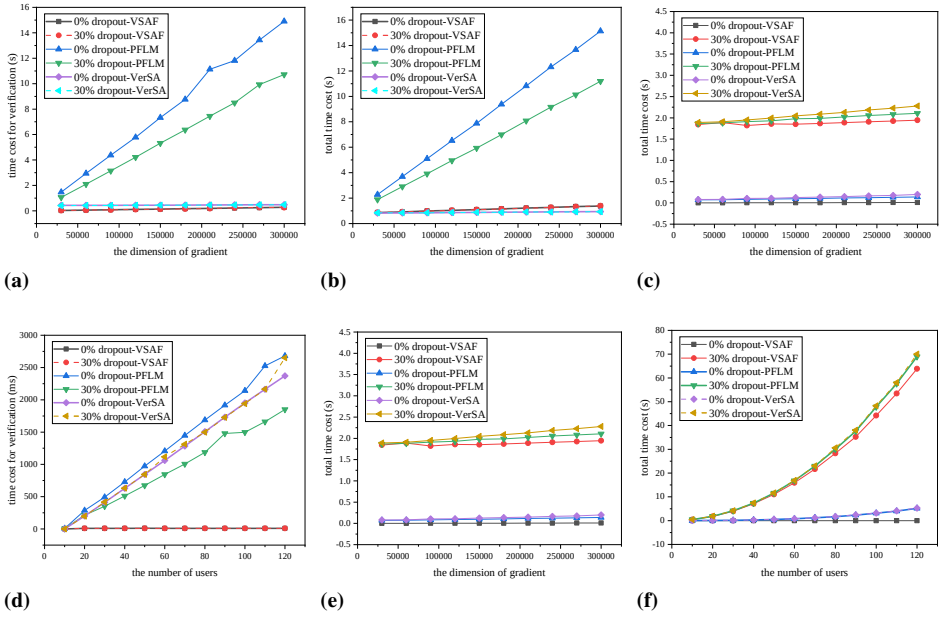

Fig. 6. The comparison of our scheme VSAF and PFLM [13] and VerSA [9] in terms of time cost for verification and total time cost. (a), (b), (c) compare the three schemes under 0% and 30% dropout rates, showing the time cost varies with gradient dimension. Specifically, (a) illustrates the time cost for verification of each user per iteration, (b) illustrates the total time cost for each user per iteration, and (c) illustrates the total time cost for CS at each iteration. (d), (e), (f) compare the three schemes under 0% and 30% dropout rates, showing the time cost varies with the number of users. Specifically, (d) illustrates the time cost for verification of each user per iteration, (e) illustrates the total time cost of each user per iteration, and (f) illustrates the total time cost of CS at each iteration

schemes in [8, 9, 13, 29] are mainly based on the double masking protocol, so the computation overhead of these schemes mainly comes from generating secret shares, masks, and ciphertexts.

[29] designs a verification scheme based on a homomorphic hash function and pseudo-random technology. The computation overhead for each user and server is $O(nd+n)$ and $O(n^2 + nd + d)$, respectively. In [9], the proposed scheme of the computation overhead for verification lies in generating secret shares, masks, and proofs. Therefore, the computation overhead is $O(nd+n)$ for each user and $O(n^2+nd)$ for the server. [13] proposes a verification scheme based on a variant of ElGamal encryption, the computation overhead amounts to $O(n^2 + nd + n)$ for each user and $O(n^2d)$ for the server. Using homomorphic hash technology and commitment scheme, [8] develops a verification scheme. The computation overhead of this scheme is $O(nd+n+d)$ for each user and $O(n^2+nd)$ for the server. [40] combines a variant of double masking protocol and differential privacy to design a privacy-preserving scheme, and uses a linear homomorphic hash to design a verification scheme. Thus, the computation overhead for each user is $O(nd + n + d)$, while for the server, it is $O(nd)$.

7.2. Experimental Settings

We implemented a prototype of our scheme, VSAF, by Python 3 and Charm-Crypto. Since our encryption scheme is based on MPC (Multi-Party Computation), the model accuracy is unaffected. Therefore, we set the gradient data in the same way as PFLM [13], randomly selecting data from a normal distribution $N(50, 20)$. We implemented the DH protocol based on the discrete logarithm problem. Then, using this DH protocol and Shamir's (t, N)-threshold secret sharing protocol, we implemented the relevant parts of our VSAF. In our scheme, we also implemented a linear homomorphic hash function for verification using the charm-crypto library. This function is irreversible and does not leak gradient information. Every experiment was conducted on a 64-bit Ubuntu OS 20.04.6 version, equipped with an Intel i3-10105 CPU and 4GB memory.

7.3. Experimental Results

Since VerifyNet is a classic secure and verifiable FL scheme, PFLM [13] and VerSA [9] are more representative than the other four masking-based schemes. Therefore, we compare the simulation experimental results of the proposed scheme with the PFLM and VerSA. We analyzed the performance of these three schemes by continuously adjusting the number of participating users N, the gradient dimension d, and the users' dropout rate.

Communication Overhead. In our scheme, the communication overhead for verification of each user mainly consists of the following two aspects: (1) In the Generation and Uploading of Ciphertexts and Proofs phase (Round 3), the user must upload proof of the gradient. (2) In the Aggregation and Verification of Parameters phase (Round 4), each user must receive a cryptographic proof from the TA validating the aggregated result. In Fig.4(a) (b) (c) (d), these subfigures illustrate the communication overhead varies with the gradient dimension and we set the number of users $N = 20$, with a threshold of

$t = \lfloor \frac{N}{2} \rfloor + 1$, *i.e.*, $t = 11$. In Fig.4(e) (f) (g) (h), these subfigures illustrate the variation in communication overhead with the number of users. and we set gradient dimension $d = 10000$ and threshold $t = \lfloor \frac{N}{2} \rfloor + 1$.

In Fig.4(a) (b), in both the PFLM [13] and VerSA [9] schemes, we see that, for both users and CS, the communication overhead for verification linearly varies with the dimension of the gradient. However, in the case of the VSAF scheme, the communication overhead for verification does not vary with the gradient dimension. This is due to our verification scheme utilizing a linear homomorphic hash function. It should be noted that the function we designed can compress d-dimensional data into one-dimensional data, thereby achieving independence of the communication overhead for verification from the gradient dimension.

As shown in Fig.4(e), in both the PFLM [13] and VerSA [9] schemes, both users and CS, the communication overhead for verification grows directly with the number of users. We can see that the communication overhead for verification of our VSAF almost remains constant regardless of the number of users. The reason is that, in PFLM and VerSA, each user needs to process the proofs of other users, which will result in the number of users affecting each user's communication. However, the users of our scheme only send their verification proofs and receive the aggregated verification proofs, so the change in the number of users has no impact on the per-user communication overhead for verification. Thus, our VSAF realizes that the verification's communication overhead for users does not depend on the number of users.

As we can see from Fig.4(a) (e), whether the dropout rate is 0% or 30%, the communication overhead of users for verification does not significantly change with the variation of the dropout rate. In VSAF, the users only send their verification proofs and receive the aggregated verification proofs, so the change in the dropout rate does not affect each user's communication overhead for verification. Thus, our proposed VSAF ensures the independence of users' communication overhead for verification from the dropout rate.

From Fig.4, we can see that the communication overhead of VSAF is more negligible than the PFLM [13] and VerSA [13]. Both PFLM [13] and VerSA [9] use a double mask scheme for privacy protection, while we use a single masking scheme, which reduces the users' communication overhead by $O(n)$. In addition, in PFLM [13], each user is required to receive other users' proofs, while in our scheme, the users do not need to receive the proofs of other users. Thus, the users of our scheme have an additional $O(n)$ reduction in communication overhead compared to PFLM [13].

Computation Overhead In Fig.5 and Fig.6, for sub-figures (a)(b)(c), we set the number of users N = 20, and the threshold $t = \lfloor \frac{N}{2} \rfloor$+1, that is, $t = 11$. For sub-figures (d)(e)(f), we set the gradient dimension d = 10000, and the threshold $t = \lfloor \frac{N}{2} \rfloor$+1.

As shown in Fig.5(d), each user's time cost for verification of these schemes does not significantly vary with a rise in the number of users. In our scheme, users only need to generate verification proofs of their own gradients through a homomorphic hash function, without requiring other users' data. Hence, the user's computation overhead for verification of our scheme should be independent of the number of users. Furthermore, as we can see from Fig.5(a) (d), the time cost for verification per user does not change with the variation of the dropout rate. Since the dropout rate reflects the changes in the number of users, and each user's computation overhead for verification is independent of the number

of users, the computation overhead for verification of each user is also independent of the dropout rate.

In Fig.5(b) (e) (c) (f), it is clear that the total time cost for users and CS increases with the growth in either the gradient dimension or the number of users. The reason is that, as the gradient dimension and the number of users increases, the data volume that users and CS need to process also increases, leading to an increase in total time cost. Furthermore, in these four sub-figures, the total time cost for users or CS increases as the dropout rate increases. This is due to the fact that our scheme uses the Shamir secret-sharing technique, and a rise in dropout users results in more secrets needing recovery, thereby increasing the total time cost. Additionally, as in sub-figures (b) (e), the changes in user time cost are not evident with the increases in the dropout rate, because we use a single masking scheme that is improved from the double mask scheme. In our scheme, online users only need to decrypt the shares' ciphertext of the dropout users and send it to CS; unlike the double mask scheme, where online users need to decrypt the ciphertext of all users.

From Fig.6, compared with the VerSA [9] and PFLM [13] schemes, it is clear that our VSAF is superior in terms of computation overhead, because our scheme is more lightweight than VerSA and PFLM concerning privacy protection and verification. In terms of privacy protection, VerSA [9] and PFLM [13] use a double mask scheme, while our VSAF uses a lightweight single masking scheme improved from the double mask scheme. Furthermore, in terms of verification, PFLM [13] uses Identity-Based Aggregate Signature technology and a variant of ElGamal encryption, which will generate a lot of time cost. VerSA [9] still designs a verification scheme based on the double mask, while VSAF uses a linear homomorphic hash function to construct a lightweight verification scheme and outsources the aggregation process of verification proofs to a third party that does not collude with CS, thereby reducing the verification cost.

Defending Against Gradient Reconstruction Attacks As shown in Fig.7, both our VSAF scheme and the FedAvg scheme [17] were trained via federated learning on the MNIST and CIFAR-10 datasets, respectively. During training, we applied a Gradient Reconstruction Attack to each method. The results indicate that under this attack, the FedAvg scheme was able to recover image information after approximately 70 rounds on MNIST and around 210 rounds on CIFAR-10. In contrast, our VSAF scheme did not leak any image information.

8. Concluding Remarks and Future Work

In this work, we propose VSAF, a verifiable and secure aggregation scheme designed for federated learning in edge computing. VSAF employs a combination of single masking with Bloom filtering for lightweight, dropout-tolerant privacy protection of user gradients. A linear homomorphic hash function is used to design a verification algorithm that ensures correct aggregation while minimizing verification overhead. Security analysis confirms the high security and correctness of VSAF, supported by comprehensive theoretical and experimental results.

In future work, we aim to reduce computational and communication overheads whilst ensuring robustness to user dropouts. In addition, we seek to develop methods to correct erroneous aggregated results while verifying their correctness.

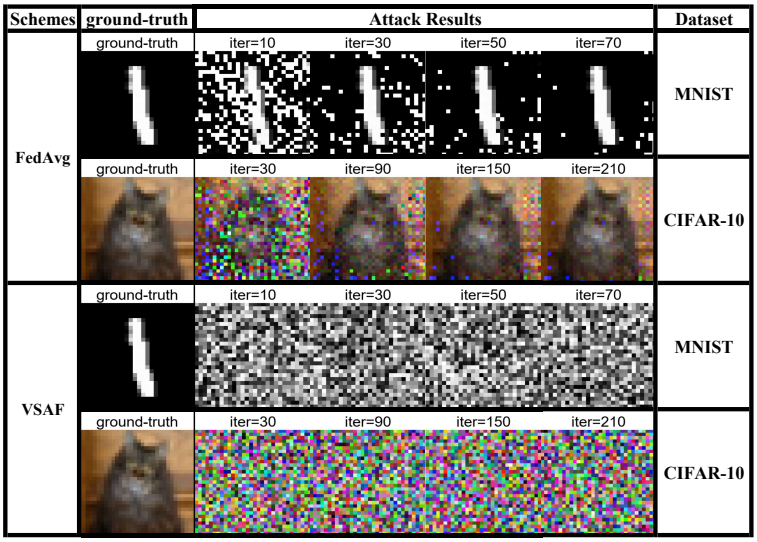

Fig. 7. Defending against gradient reconstruction attacks

Acknowledgments. This work is supported in part by the Scientific Research Fund of Hunan Provincial Education Department (No. 24A0337), the Natural Science Foundation of Hunan Province (No. 2025JJ50348), and the Natural Science Foundation of Fujian Province (No. 2022J05106). The authors appreciate the editor and anonymous reviewers for their invaluable feedback.

References

1. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep learning with differential privacy. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 308–318 (2016)
2. Bell, J.H., Bonawitz, K.A., Gascón, A., Lepoint, T., Raykova, M.: Secure single-server aggregation with (poly) logarithmic overhead. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1253–1269 (2020)
3. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (1970)
4. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1175–1191 (2017)
5. Chen, Y., Qin, X., Wang, J., Yu, C., Gao, W.: Fedhealth: A federated transfer learning framework for wearable healthcare. IEEE Intelligent Systems 35(4), 83–93 (2020)
6. Cisco: Cisco annual internet report (2018–2023) (Mar 2020), https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf
7. Filho, C.P., Marques Jr, E., Chang, V., Dos Santos, L., Bernardini, F., Pires, P.F., Ochi, L., Delicato, F.C.: A systematic literature review on distributed machine learning in edge computing. Sensors 22(7), 2665 (2022)

8. Guo, X., Liu, Z., Li, J., Gao, J., Hou, B., Dong, C., Baker, T.: Verifl: Communication-efficient and fast verifiable aggregation for federated learning. IEEE Transactions on Information Forensics and Security 16, 1736–1751 (2020)
9. Hahn, C., Kim, H., Kim, M., Hur, J.: Versa: Verifiable secure aggregation for cross-device federated learning. IEEE Transactions on Dependable and Secure Computing (2021)
10. Han, G., Zhang, T., Zhang, Y., Xu, G., Sun, J., Cao, J.: Verifiable and privacy preserving federated learning without fully trusted centers. Journal of Ambient Intelligence and Humanized Computing pp. 1–11 (2022)
11. Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., Ramage, D.: Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604 (2018)
12. Hitaj, B., Ateniese, G., Perez-Cruz, F.: Deep models under the gan: information leakage from collaborative deep learning. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. pp. 603–618 (2017)
13. Jiang, C., Xu, C., Zhang, Y.: Pflm: Privacy-preserving federated learning with membership proof. Information Sciences 576, 288–311 (2021)
14. van Leeuwen, J. (ed.): Computer Science Today. Recent Trends and Developments, Lecture Notes in Computer Science, vol. 1000. Springer-Verlag, Berlin Heidelberg New York (1995)
15. Li, T., Li, J., Chen, X., Liu, Z., Lou, W., Hou, Y.T.: Npmml: A framework for non-interactive privacy-preserving multi-party machine learning. IEEE Transactions on Dependable and Secure Computing 18(6), 2969–2982 (2020)
16. Lin, L., Zhang, X.: Ppverifier: A privacy-preserving and verifiable federated learning method in cloud-edge collaborative computing environment. IEEE Internet of Things Journal 10(10), 8878–8892 (2022)
17. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.y.: Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR (20–22 Apr 2017), https://proceedings.mlr.press/v54/mcmahan17a.html
18. Melis, L., Song, C., De Cristofaro, E., Shmatikov, V.: Inference attacks against collaborative learning. arXiv preprint arXiv:1805.04049 13 (2018)
19. Mou, W., Fu, C., Lei, Y., Hu, C.: A verifiable federated learning scheme based on secure multiparty computation. In: International Conference on Wireless Algorithms, Systems, and Applications. pp. 198–209. Springer (2021)
20. Phong, L.T., Aono, Y., Hayashi, T., Wang, L., Moriai, S.: Privacy-preserving deep learning via additively homomorphic encryption. IEEE Transactions on Information Forensics and Security 13(5), 1333–1345 (2018)
21. Ribičre, M., Charlton, P.: Ontology overview. Motorola Labs, Paris (2002), [Online]. Available: http://www.fipa.org/docs/input/f-in-00045/f-in-00045.pdf (current October 2003)
22. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (nov 1979), https://doi.org/10.1145/359168.359176
23. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. pp. 1310–1321 (2015)
24. Song, C., Ristenpart, T., Shmatikov, V.: Machine learning models that remember too much. In: Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security. pp. 587–601 (2017)
25. Treleaven, P., Smietanka, M., Pithadia, H.: Federated learning: the pioneering distributed machine learning and privacy-preserving data technology. Computer 55(4), 20–29 (2022)
26. Vailshery, L.S.: Internet of things (iot)—statistics and facts. https://www.statista.com/study/27915/internet-of-things-iot-statista-dossier/ (2021)

27. Wang, F., He, Y., Guo, Y., Li, P., Wei, X.: Privacy-preserving robust federated learning with distributed differential privacy. In: 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 598–605 (2022)
28. Wang, X., Bettini, C., Brodsky, A., Jajoida, S.: Logical design for temporal databases with multiple granularities. ACM Transactions on Database Systems 22(2), 115–170 (1997)
29. Xu, G., Li, H., Liu, S., Yang, K., Lin, X.: Verifynet: Secure and verifiable federated learning. IEEE Transactions on Information Forensics and Security 15, 911–926 (2019)
30. Xu, G., Li, H., Zhang, Y., Xu, S., Ning, J., Deng, R.H.: Privacy-preserving federated deep learning with irregular users. IEEE Transactions on Dependable and Secure Computing 19(2), 1364–1381 (2020)
31. Yan, Y., Kamel, M.B., Zoltay, M., Gál, M., Hollós, R., Jin, Y., Péter, L., Tényi, Á.: Fedlabx: a practical and privacy-preserving framework for federated learning. Complex & Intelligent Systems pp. 1–14 (2023)
32. Zhang, J., Liu, Y., Wu, D., Lou, S., Chen, B., Yu, S.: Vpfl: A verifiable privacy-preserving federated learning scheme for edge computing systems. Digital Communications and Networks 9(4), 981–989 (2023)
33. Zhang, S., He, J., Liang, W., Li, K.: Mmds: A secure and verifiable multimedia data search scheme for cloud-assisted edge computing. Future Generation Computer Systems 151, 32–44 (2024), https://www.sciencedirect.com/science/article/pii/S0167739X23003564
34. Zhang, S., Hu, B., Liang, W., Li, K.C., Gupta, B.B.: A caching-based dual k-anonymous location privacy-preserving scheme for edge computing. IEEE Internet of Things Journal 10(11), 9768–9781 (2023)
35. Zhang, S., Hu, B., Liang, W., Li, K.C., Pathan, A.S.K.: A trajectory privacy-preserving scheme based on transition matrix and caching for iiot. IEEE Internet of Things Journal pp. 1–1 (2023)
36. Zhang, S., Yan, Z., Liang, W., Li, K.C., Dobre, C.: Baka: Biometric authentication and key agreement scheme based on fuzzy extractor for wireless body area networks. IEEE Internet of Things Journal pp. 1–1 (2023)
37. Zhang, S., Yang, Y., Liang, W., Sandor, V.K.A., Xie, G., Raymond, K.K.: Mkss: An effective multi-authority keyword search scheme for edge–cloud collaboration. Journal of Systems Architecture 144, 102998 (2023), https://www.sciencedirect.com/science/article/pii/S1383762123001777
38. Zhang, Y., Yu, H.: Towards verifiable federated learning. arXiv preprint arXiv:2202.08310 (2022)
39. Zhao, J., Zhu, H., Wang, F., Lu, R., Liu, Z., Li, H.: Pvd-fl: A privacy-preserving and verifiable decentralized federated learning framework. IEEE Transactions on Information Forensics and Security 17, 2059–2073 (2022)
40. Zhou, H., Yang, G., Huang, Y., Dai, H., Xiang, Y.: Privacy-preserving and verifiable federated learning framework for edge computing. IEEE Transactions on Information Forensics and Security 18, 565–580 (2023)
41. Zhu, L., Liu, Z., Han, S.: Deep leakage from gradients. Advances in neural information processing systems 32 (2019)

Shiwen Zhang received his Ph.D. degree from the College of Computer Science and Electronic Engineering, Hunan University, in 2016. He is currently an associate professor at the School of Computer Science and Engineering, Hunan University of Science and Technology. He is a member of IEEE and CCF. His research interests include identity authentication, security and privacy issues in Wireless Body Area Networks (WBAN), cloud computing, privacy protection, and information security.
Email: shiwenzhang@hnust.edu.cn

Feixiang Ren received a B.S. degree from Henan University of Technology, and currently pursuing an M.S. degree from the school of Computer Science and Engineering at Hunan University of Science and Technology. His research interests include privacy-preserving and verifiable federated learning. Email: rfx.point@mail.hnust.edu.cn

Wei Liang received a Ph.D. degree in computer science and technology from Hunan University in 2013. He was a Post-Doctoral Scholar at Lehigh University from 2014 to 2016. He is currently a Professor and the Dean of the School of Computer Science and Engineering at Hunan University of Science and Technology, China. He has authored or co-authored more than 140 journal and conference papers. His research interests include identity authentication in WBAN and security management in wireless sensor networks (WSN). Email: weiliang@hnu.edu.cn

Kuanching Li is a Professor at the School of Computer Science and Engineering, Hunan University of Science and Technology. Dr. Li has co-authored over 150 conference and journal papers, holds several patents, and serves as an associate and guest editor for various scientific journals. He has also held chair positions at several prestigious international conferences. His research interests include cloud and edge computing, big data, and blockchain technologies. Dr. Li is a Fellow of the Institution of Engineering and Technology (IET). Email: aliric@hnust.edu.cn

Al-Sakib Khan Pathan received a B.Sc. degree in computer science and information technology from the Islamic University of Technology (IUT), Bangladesh, in 2003 and a Ph.D. degree in computer engineering from Kyung Hee University, South Korea, in 2009. He is a Professor at the Department of Computer Science and Engineering, United International University (UIU), Bangladesh. He has served as General Chair, Organizing Committee Member, and TPC member in top-ranked conferences such as INFOCOM, CCGRID, GLOBECOM, ICC, LCN, GreenCom, and IRI. He received the IEEE Outstanding Leadership Award (GreenCom'13) and two IEEE Outstanding Service Awards (IRI'20, IRI'21), and has co-edited/co-authored 34 books. Email: sakib.pathan@gmail.com

Received: February 21, 2025; Accepted: April 26, 2025.