entropy

MDPI

Article

Communication-Efficient and Privacy-Preserving Verifiable Aggregation for Federated Learning

Kaixin Peng [1], Xiaoying Shen [2,3,*], Le Gao [1,*], Baocang Wang [2] and Yichao Lu [1]

[1] The Faculty of Intelligent Manufacturing, Wuyi University, Jiangmen 529020, China; kxinpeng@outlook.com (K.P.); lkiolyc@outlook.com (Y.L.)
[2] The State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China; bcwang@xidian.edu.cn
[3] The Key Laboratory of Cryptography of Zhejiang Province, Hangzhou Normal University, Hangzhou 311121, China
* Correspondence: shenxiaoying@xidian.edu.cn (X.S.); le.gao@nscc-gz.cn (L.G.)

Abstract: Federated learning is a distributed machine learning framework, which allows users to save data locally for training without sharing data. Users send the trained local model to the server for aggregation. However, untrusted servers may infer users' private information from the provided data and mistakenly execute aggregation protocols to forge aggregation results. In order to ensure the reliability of the federated learning scheme, we must protect the privacy of users' information and ensure the integrity of the aggregation results. This paper proposes an effective secure aggregation verifiable federated learning scheme, which has both high communication efficiency and privacy protection function. The scheme encrypts the gradients with a single mask technology to securely aggregate gradients, thus ensuring that malicious servers cannot deduce users' private information from the provided data. Then the masked gradients are hashed to verify the aggregation results. The experimental results show that our protocol is more suited for bandwidth-constraint and offline-users scenarios.

Keywords: federated learning; privacy protection; verifiability; homomorphic hash function

Citation: Peng, K.; Shen, X.; Gao, L.; Wang, B.; Lu, Y. Communication-Efficient and Privacy-Preserving Verifiable Aggregation for Federated Learning. *Entropy* **2023**, *25*, 1125. https://doi.org/10.3390/e25081125

Academic Editors: Eirik Rosnes and Hsuan-Yin Lin

Received: 16 May 2023
Revised: 21 July 2023
Accepted: 24 July 2023
Published: 27 July 2023

1. Introduction

Federated learning (FL) [1–3] is a distributed machine learning framework proposed by Google for privacy protection. In this framework, the involved users can conduct independent training and collaborate with the central server to obtain a high-performance shared global model. Unlike traditional distributed machine learning [4,5], users train their models locally instead of directly sharing private data. The server aggregates users' trained local parameters or gradients rather than training on their data. Therefore, FL addresses concerns related to data privacy, data ownership, and data silos [6–8].

However, in the federated learning framework, the central server is generally considered untrusted and may infer the private data of users. Recent research [9–11] shows that adversaries can directly reconstruct users' original data from the shared gradients. This means that an unreliable server can easily extract users' private information from the transmitted gradients. Additionally, a malicious central server driven by self-interest may return incorrect results to users to conserve computing resources [12,13], leading to increased training iterations and a low-precision global model. Hence, protecting user privacy and ensuring data integrity are critical concerns within federated learning training protocols.

To address the above issues, Xu et al. [14] proposed a privacy-preserving and verifiable federated learning approach using secret sharing and homomorphic signatures. However, as the dimension of the vector increases, this method will greatly increase communication

costs. The method proposed in reference [15] can efficiently validate results using commitment techniques and linear homomorphic hashing. However, it requires users to share hash and commitment values, resulting in additional communication overhead. Furthermore, ref. [16] introduced a model recovery attack on shared gradients hashes and suggested using shared vectors for verification. Nonetheless, the direct sharing of a verification vector can impose a significant communication burden on resource-constrained devices such as edge devices and sensors.

In this paper, we propose a communication-efficient and privacy-preserving verifiable aggregation federated learning protocol to facilitate training on limited bandwidth devices. Specifically, we utilize a single mask mechanism [17] to encrypt the gradients, ensuring privacy-preserving gradients aggregation. Additionally, we design a verification method to authenticate the integrity of the aggregated results against malicious server attacks. Our contributions can be summarized as follows:

- We propose a communication-efficient and privacy-preserving aggregation framework for the federated training process. By employing the additive homomorphism property of secret sharing, we protect the privacy of user information, while a homomorphic hash function enables verification.
- We devise a novel strategy to counter model recovery attacks. By employing homomorphic hashing on the masked gradients rather than directly hashing the gradients, we effectively protect users' privacy against model recovery attacks on gradients hashes.
- We give a comprehensive security analysis to prove the high security of our scheme. Besides, extensive experiments demonstrate that our scheme has high communication efficiency with a moderate increase in computational costs.

The rest of the paper is organized as follows: we provide related work in Section 2, introduce preliminaries in Section 3, and describe the system model of our scheme in Section 4. After that, we discuss the construction of our scheme in detail in Section 5. The security analysis is provided in Section 6. Experiment evaluation of the proposed scheme is presented in Section 7. Finally, we conclude our paper in Section 8. Additionally, the abbreviations are used in the paper as follows.

2. Related Work

2.1. Privacy-Preserving FL

There have been many studies on federated learning about privacy protection. These studies primarily adopt the following three techniques: differential privacy (DP) [8,18–20], homomorphic encryption (HE) [9,21,22] and secure multiparty computing (SMC) [23–28].

In DP, Abadi et al. [18] proposed a training algorithm for differential privacy protection, which can obtain a more rigorous estimate of total privacy loss through quantitative analysis of privacy. Zhou et al. [19] proposed a differential privacy federated learning approach for detecting and filtering anomalous parameters uploaded from malicious terminal devices through edge node validation. However, methods based on differential privacy can cause a loss of global model accuracy owing to the introduction of random noise. In HE, Phong et al. [9] proposed a privacy-preserving deep learning model based on additive HE, which can realize high-precision model training in asynchronous federated learning. Park et al. [21] proposed a distributed homomorphic cryptosystem for implementing a privacy-preserving federated learning scheme. However, in the HE methods, the use of the same key pair by all users can lead to severe information leakage if users collude with the server. Additionally, the computation and transmission costs of ciphertexts are significant and may not be suitable for scenarios with resource-constrained devices that may dropout.

To address the privacy and dropout concerns on resource-constrained devices, some federated learning approaches have been proposed. Zhao et al. [29] characterized the optimal communication cost for the information-theoretic secure aggregation problem. Bonawitz et al. [23] designed a double-mask mechanism using SMC techniques for implementing privacy-preserving federated learning. To mitigate the supplementary overhead

introduced by handling dropped users' seeds in [23], So et al. [27] employed a one-shot aggregate-mask reconstruction approach to achieve secure aggregation in federated learning. Jahani-Nezhad et al. [30] proposed a secure aggregation protocol for federated learning systems that reduces communication overhead without compromising security. Schlegel et al. [28] proposed two privacy-preserving federated learning schemes to mitigate the impact of user dropout. Du et al. [22] proposed a threshold multi-key homomorphic encryption scheme to achieve secure, robust, and efficient federated learning training. Lu et al. [24] designed a secure aggregation protocol called Top-k SparseSecAgg to reduce communication cost and training time.

2.2. FL with Verifiable Computation

Many works have been proposed in verifiable federated learning. Xu et al. [14] proposed a privacy-preserving and verifiable federated learning scheme named VerifyNet. They used bilinear pairing and hash functions to achieve results verifiability, and used shamir secret sharing to achieve secure aggregation of gradients. However, the protocol requires users to use zero-knowledge proofs, and its communication cost increases linearly with the number of gradients. Furthermore, as mentioned in the attacks discussed in [16], the hashes allow malicious servers to infer users' private data. Guo et al. [15] proposed VerifL, a communication-efficient and fast verifiable aggregation federated learning scheme, which achieved the verification communication independent of vector dimension. They utilized linearly homomorphic hash and commitment techniques to verify the aggregation results. However, it is vulnerable to the same attacks as mentioned in [16]. Zhang et al. [31] proposed a privacy-preserving and verifiable federated learning scheme that used Paillier encryption for gradients protection and bilinear aggregation signature for verification of the aggregation results. However, since users use the same encryption and decryption key, user privacy may be leaked if a malicious user colludes with the server. In the bilinear aggregation signature verification scheme, users generate signatures of gradients and send them to the server for aggregation. Users then verify whether the aggregated signature is equal to their own signature. However, a lazy server can aggregate only partial user gradients and signatures and send them to users for verification, which can pass the verification process. Additionally, the bilinear aggregation signature has a high verification cost.

Based on this, this paper aims to propose an aggregation verifiable federated learning method with efficient communication and privacy protection. Firstly, our approach can protect users' private data against attacks from malicious servers and colluding users. On the other hand, in scenarios where bandwidth is extremely limited, our approach has a small communication overhead. Specifically, we compare the features of our approach with other existing solutions in Table 1.

Table 1. Functionality comparison.

Scheme	Privacy-Preserving	Verifiable	Drop-Tolerant	Resist Model Recovery Attacks	Communication-Efficient
PPML [23]	✓	-	✓	-	×
PPDL [9]	✓	-	×	-	×
VerifyNet [14]	✓	✓	✓	×	×
VeriFL [15]	✓	✓	✓	×	✓
Versa [16]	✓	✓	✓	✓	×
Ours	✓	✓	✓	✓	✓

3. Preliminaries

3.1. Federated Learning

Federated learning is a distributed machine learning framework. A server and multiple users work together to train a model. Suppose there are n users and a server in federated learning, and each user locally saves the dataset $D_i = \{(d_{ij}, o_{ij}), j = 1, 2, \ldots, J\}$.

Each user downloads the latest global model w_{t-1} from the server during an iteration. The loss function $L_f^i(D_i, w) = \frac{1}{|D_i|}\sum_{(d_{ij},o_{ij})\in D_i} l\left(o_{ij}; f\left(d_{ij}, w\right)\right)$ is calculated by using the local dataset. $|D_i|$ represents the size of D_i. $f(d_{ij}, w)$ denotes the predicted result and w denotes model parameters. $l(o_{ij}; f(d_{ij}, w))$ represents the loss result obtained by predicting the sample (d_{ij}, o_{ij}) on the given model parameters. To expedite model training, stochastic gradient descent (SGD) is utilized for computing the gradients $x_i = \nabla L_f^i\left(D_i^k, w\right)$. ∇L_f^i represents the derivative of the loss function and D_i^k represents a subset of the dataset D_i. Each user sends the calculated gradients x_i to the server. The server calculates an updated global model, i.e., $w_t \leftarrow w_{t-1} - \alpha\sum_{i=1}^{n}\frac{|D_i|}{|D|}x_i$, where $|D| = \sum_{i=1}^{n}|D_i|$. The server then broadcasts the aggregated global model w_t again, repeating the process until the global model converges or a certain number of iterations is reached.

3.2. Secret Sharing

Shamir secret sharing [32] divides a secret s into n secret shares, and only when the secret shares are greater than or equal to t can the secret s be reconstructed. For a Shamir (t, n) threshold scheme, let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of shared key users. In our scheme, u_i represents the index number uniquely identifying the user. User u_s represents a secret holder. The scheme is as follows:

- **SS.Share** (s, t, U): For the secret $s \in GF(q), q > n$, the user u_s selects any $t-1$ positive integers to construct a polynomial $f(x) = s + a_1x + a_2x^2 + \ldots + a_{t-1}x^{t-1}(\text{mod } q)$. The user then calculates $f(u_1), f(u_2), \ldots, f(u_n)$, and sends $\{u_i, f(u_i)\}$ to user u_i.
- **SS.Recon**$(\{u_i, f(u_i)\}_{u_i\in U}, t)$: Given any t of $\{u_i, f(u_i)\}$, the secret holder can calculate the coefficients $s, a_1, \ldots, a_{t-1}$ of the polynomial $f(x)$ using Lagrange interpolation and obtain the secret s. To reduce the computation, we adopt a simpler method of calculating the secret sum: $s = \sum_{j=0}^{t} f(u_j)\prod_{p=0,p\neq j}^{t}\frac{u_p}{u_p-u_j}$.

Shamir secret sharing satisfies additive homomorphism [33]. For example, suppose two users, u_1 and u_2, possess secrets s_1 and s_2, respectively. To secretly share the secret s_1 among n users in U, user u_1 employs the function **SS.Share** (s_1, t, U) to select a polynomial $f(x) = s_1 + a_1x + a_2x^2 + \ldots + a_{t-1}x^{t-1}(\text{mod } q)$ and calculates n shares of s_1, denoted as $\{u_i, f(u_i)\}$, which are sent to the corresponding users u_i. Similarly, user u_2 uses the function **SS.Share** (s_2, t, U) to select a polynomial $g(x) = s_2 + b_1x + b_2x^2 + \ldots + b_{t-1}x^{t-1}(\text{mod } q)$, and distributes shares $\{u_i, g(u_i)\}$ to the users u_i. Consequently, each user u_i can locally compute $\{u_i, (f(u_i) + g(u_i))\}$. Then, in collaboration with other users where the number of users involved is greater than t, the secret $s_1 + s_2$ can be reconstructed using the function **SS.Recon** $(\{u_i, (f(u_i) + g(u_i))\}_{u_i\in U}, t)$. Similarly, it can implement shared refactoring of any number of secrets.

3.3. Homomorphic Hash Function

A homomorphic hash function [34] includes three algorithms: **HHF.Gen**, **HHF.Hash** and **HHF.Eval**.

- **HHF.Gen** $(1^k, 1^d)$: This algorithm initializes the system parameters by entering a random parameter k and a dimension d. The algorithm outputs common parameters pp, including the elliptic curve group G of order q, its generator g and d distinct elements $g_1, g_2, \ldots, g_m \in G$.
- **HHF.Hash** (x): Input a m dimension vector x, the algorithm will output the hash value of $x : h \leftarrow \prod_{i=1}^{m} g_i^{x[i]} \in G$.
- **HHF.Eval** $(h_1, h_2, \ldots, h_k)$: Input k hash values, this algorithm calculates the combination of hashes $h \leftarrow \prod_{i=1}^{k} h_i$. It satisfies the equation $h =$ **HHF.Hash**$(x_1 + x_2 + \ldots + x_k)$.

Collision-resistant: **HHF** is said to be collision-resistant if there is no adversary $\mathcal{A}$ that can satisfy the following experiment with non-negligible advantage.

$$\mathrm{Adv}^{\mathrm{coll}}_{\mathcal{A},\mathrm{HHF}}(\mathrm{k}) := \Pr\left[\mathbf{HHF.Hash}(\mathrm{x}) = \mathbf{HHF.Hash}(\mathrm{x}') \quad : \quad \begin{array}{c} \mathrm{pp} \leftarrow \mathbf{HHF.Gen}\left(1^{\mathrm{k}}, 1^{\mathrm{d}}\right) \\ (\mathrm{x}, \mathrm{x}') \leftarrow \boldsymbol{A}(\mathrm{pp}) \end{array} \right].$$

One-way: **HHF** is said to be one-way, if for any PPT adversary $\mathcal{A}$ that can satisfy the following experiment with non-negligible advantage for the security parameter k and the vector dimension m.

$$\mathrm{Adv}^{\mathrm{one}}_{\mathcal{A},\mathrm{HHF}}(\mathrm{k}) := \Pr\left[\mathrm{x} = \mathrm{x}' : \begin{array}{c} \mathrm{pp} \leftarrow \mathbf{HHF.Gen}\left(1^{\mathrm{k}}, 1^{\mathrm{d}}\right) \\ \mathrm{h} \leftarrow \mathbf{HHF.Hash}(\mathrm{x}) \\ \mathrm{x}' \leftarrow \mathcal{A}(\mathrm{pp}, \mathrm{h}) \end{array} \right].$$

3.4. Key Agreement

In our scheme, Diffie-Hellman (DH) [35] key agreement protocols allow any two users to negotiate a key. This protocol consists of three algorithms: **KA.Setup**, **KA.Gen** and **KA.Agree**.

- **KA.Setup** (1^d): The algorithm inputs a security parameter k and outputs a public parameter pp.
- **KA.Gen** (pp): The algorithm will output a key pair (pk_i, sk_i) for user u_i as input pp.
- **KA.Agree** (sk_i, pk_j): This algorithm inputs the private key sk_i of user u_i and the public key pk_j of user u_j, and output an agreed key s_{ij}.

3.5. Authenticated Encryption

Authenticated encryption [36] includes three algorithms:

- **AE.Gen** (pp): The algorithm inputs a security parameter pp and outputs the secret symmetric key k.
- **AE.Enc** (k, m): The algorithm inputs a symmetric key k and a message m, outputs ciphertext c.
- **AE.Dec** (k, c): The algorithm inputs a symmetric key k and a ciphertext c, outputs plaintext m of c.

4. System Model

In this section, we first describe our system framework and communication model, introduce the threat model of our scheme, and define the design goals.

4.1. System Framework and Communication Model

We focus on how multiple users collaborate with the server to train a model with good performance. Each user is a mobile device that stores a dataset and can connect to the server. Our model consists of three entities: a trusted third party, a central server and users, as shown in Figure 1.

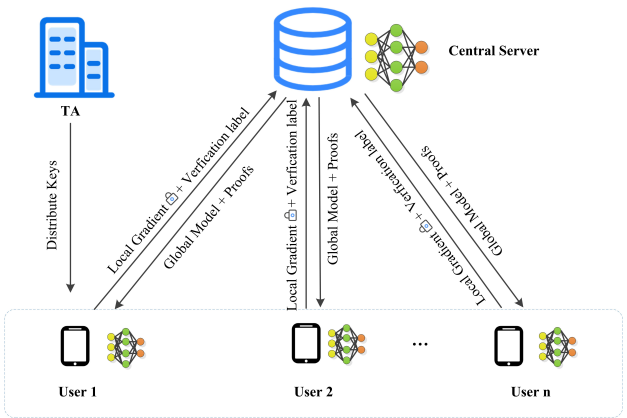

Figure 1. The system model of our scheme.

Trusted third party (TA): TA is mainly responsible for parameter initialization. It generates public and private key pairs and system parameters for each user participating in the training. Then it distributes these key pairs to the corresponding users and forward the general parameters.

User: Each user who stores the dataset locally can voluntarily choose whether to connect to the server and participate in model training. However, they may be due to network delay, system disconnection, system timeout, or active exit at any time [17]. In each iteration, users train with their local datasets to obtain the gradients and send the encrypted gradients and verification label to the server. After the completion of server-side aggregation, users are provided with the aggregated results and the corresponding verification proofs. Then, users verify the correctness of the aggregated results.

Central server: A server with sufficient computing resources which does not have a training dataset, and needs to collaborate with users to train a global model. The central server is responsible for aggregating gradients of encryption to update the worldwide model. It sends the aggregated results and verification proofs back to each user.

The protocol of our scheme is mainly divided into five rounds. Each interaction between the server and users is recorded as a round. If any round finds that the online users are less than t, the protocol stops immediately. The detailed content is as follows:

- Initialization: TA distributes system parameters to users and the public key of user will be shared.
- Round 1: User shares a secret with other online users, which will be encrypted using the public key.
- Round 2: User sends the encrypted model and the verification label to the server.
- Round 3: User produces a share which will be used for refactoring the secrets and sends it to the server. The server calculates aggregated results and aggregate verifiable labels.
- Round 4: User verifies the integrity of the aggregated results using verification proofs.

4.2. Threat Model

We adopted a threat model similar to reference [16], in which TA is a fully trusted entity that does not engage in any collusion. In this model, the server is considered malicious and attempts to deduce users' training data. In addition, the server may manipulate the aggregation results to deceive users for unlawful purposes. We assume that users are honest-but-curious entities who correctly perform the training process and upload the masked gradients and verification label. However, users may try to deduce other users' private data from the server's returned results. Moreover, we consider the potential collusion between users and the server in practical federated learning scenarios, and the number of colluding users does not exceed the threshold value t in the context of secret-sharing. We exclude the scenario where users and the server collude to pass the verification process in our work. This is because multi-client verifiable computations fail to attain verifiability when the user colludes with the server, as demonstrated in [37].

4.3. Design Goals

In the system model, our work needs to meet the following two design objectives:

- **Privacy-preserving:** Privacy preservation is an essential part of federated learning. Therefore, our scheme must ensure that users' private data can resist the adversaries' attacks.
- **Integrity verification:** In federated learning, each user's data are stored locally and trained locally. They can get a high-quality model by receiving the aggregated results returned by the server. If the server manipulates the aggregation results to deceive users, it could significantly impact their training. Therefore, our scheme ensures that the server cannot forge the aggregation results to deceive users into passing the verification process.

5. The Proposed Scheme

5.1. Overview

In this section, we provide a detailed description of the structure of our proposed solution. We will focus on addressing two issues: (1) how to securely aggregate without disclosing users' sensitive information, (2) how to perform aggregation results' integrity verification more efficiently in the context of limited bandwidth.

Before the system iteration training begins, the server will select a certain proportion of users to participate. The chosen users engage in five rounds of interactions with the server, as depicted in Figure 2. During these five rounds of interactions, users may exit the training protocol at any time due to network latency or other issues. We establish in Theorem 2 that even if some users prematurely terminate the training protocol, our proposed approach can still proceed without disruption. Each selected user possesses a unique and valid identity ID, denoted as i or j. Each user gets its local gradients after each round of training. Then each user adopts the single mask mechanism to encrypt gradients to obtain $[[x_i]]$ and hash function to calculate the verification label T_i at the same time. Each user will upload $[[x_i]]$ and T_i to the central server. After receiving the data uploaded by users, the central server aggregates the encrypted gradients and verification labels. Then it returns the aggregated results and verification proofs to users. Finally, each user will verify the correctness of the aggregation results. If the verification passes, the aggregation results will be used for the next round of local training. Otherwise, users will exit the training protocol.

5.2. Initialization

The initialization phase of the system is mainly guided by TA, which generates system parameters and public private key pairs for each user before training the model. First, TA generates public/secret keys (u_i^{pk}, u_i^{sk}) for each user and distributes them. In addition, a random seed s and hash parameters $g_1, g_2, \ldots, g_m$ are also generated and forwarded to all users. Before users start training, the server randomly generates the learning rate α and initializes the model parameter w and sends them to users. Upon receiving these parameters, each user shares its public key with other users through the server to establish a shared encryption key for exchanging secrets.

5.3. Secure Aggregation

This section aims to protect users' training model information from leakage. We use a single mask protocol to encrypt the gradients of users. Single mask mainly utilizes the homomorphism of secret sharing to achieve secure aggregation. The single mask method has stronger user dropout robustness than the double mask method.

We simply use a single mask to encrypt the gradients. Each user generates a mask r_i and calculates $y_i = x_i + r_i$. The server calculates $z = \sum_{u_i \in U} y_i = \sum_{u_i \in U} x_i + \sum_{u_i \in U} r_i$. If the server wants to get the aggregated results $\sum_{u_i \in U} x_i$, it must know the aggregated mask $R = \sum_{u_i \in U} r_i$. However, the server cannot ask users to send r_i, as this would directly disclose the values x_i. A way of the server refactoring R is to utilize the homomorphism of secret sharing $R = Share_{\sum r_i} = \sum Share_{r_i}$, so the server only needs to get all the shares of r_i to reconstruct R. User i generates n secret shares of r_i before sending the encrypted gradients. One share of r_i is represented by r_i^j, which represents the share of the secret r_i sent by user i to user j. Thus, each user can obtain r_j^i from other users. The r_j^i is only part of the information held by the user i about r_j. The server will ask the online users i to send $R^i = \sum_{u_j \in U} r_j^i$, then the server calculates $\sum_{u_j \in U} R^i = R$. Since user i only knows r_j^i but not r_j, and the server only knows R_i but not r_i, they cannot learn anything about x_i. This single mask mechanism is much more robust against user dropout. If some users drop out of the protocol during the aggregation phase, then the server does not receive any model information for these users. The server only needs to send the set of online users during the aggregation phase to each user. Each user will then aggregate the r_j^i value of the online

users set. So long as the number of users online exceeds the threshold value t, the server can reconstruct R. So, the aggregation results can be calculated correctly.

- **Round 0(Initialization)**

  TA

  - Generate the security parameter k, the public/secret keys (u_i^{pk}, u_i^{sk}) for each user i, and hash parameters $hp \leftarrow \mathbf{HHF.Gen}(1^k, 1^d)$.

  user i:

  - Send the public key (u_i^{pk}, u_i^{sk}) to the server.

  Server:

  - Receive messages from at least t users (represented as $U_1 \subseteq U$), where t is the threshold of the Shamir's t-out-of-N protocol.
  - Broadcast $\{j, u_j^{pk}\}_{u_j \in U_1}$ to each user in U_1.

- **Round 1(Key Sharing)**

  user i:

  - Receive the $\{j, u_j^{pk}\}_{u_j \in U_1}$ and ensure $U_1 \geq t$.
  - Select a random number r_i and split r_i into t-out-of-U_1 shares $\{i, r_i^j\} \leftarrow \mathbf{SS.Share}(r_i, t, U_1)$, r_i^j is the shares of r_i between user i and user j.
  - Compute $C_{i,j} \leftarrow \mathbf{AE.Enc}(\mathbf{KA.Agree}(u_i^{sk}, u_j^{pk}), i\,||j\,||r_i^j)$.
  - Send $\{C_{i,j}\}_{u_j \in U_1}$ to the server.

  Server:

  - Receive messages from at least t users, let $U_2 \subseteq U_1$ be this set of users.
  - Broadcast $\{C_{i,j}\}_{u_j \in U_2}$ to each user.

- **Round 2(Masked Input)**

  user i:

  - Receive the $\{C_{i,j}\}_{u_j \in U_2}$, Check whether $U_2 \subseteq U_1$ and ensure $U_2 \geq t$.
  - Encrypt the local gradient as $[[x_i]] = x_i + r_i + PRG(s)\ (\mathrm{mod}\ B)$.
  - Calculate $T_i = \mathbf{HHF.Hash}(x_i + r_i)$.
  - Send $\{[[x_i]], T_i\}$ to the server.

  Server:

  - Receive messages from at least t users, let $U_3 \subseteq U_2$ be this set of users.
  - Broadcast the set of U_3 to each user.

- **Round 3(Unmasking)**

  user i:

  - Check whether $U_3 \subseteq U_2$ and ensure $U_3 \geq t$.
  - Decrypt each $\{C_{i,j}\}_{u_j \in U_2}$ to $i||j||r_j^i$.
  - Calculate the sum $R^i = \sum_{u_j \in U_3} r_j^i$ and send R^i to the server.

  Server:

  - Receive R^i from more than t users, let $U_4 \subseteq U_3$ be this set of users.
  - Calculate $R = \mathbf{SS.Recon}(\{i, R^i\}_{u_i \in U_4}, t)$.
  - Calculate the aggregated gradients and verification label as $X = \sum_{u_i \in U_3} [[x_i]] - R = \sum x_i + |U_3| PRG(s), T = \prod_{i=1}^{n} T_i$.
  - Broadcast $\{X, T, R\}$ to each user.

- **Round 4(Verification)**

  user i:

  - Receive the $\{X, T, R\}$.
  - Accpet X if $\mathbf{HHF.Hash}(X - nPRG(s) + R) = T$.

Figure 2. Our protocol in detail.

As mentioned above, we have designed a simple protocol for secure summation on the server side. However, the effectiveness of our approach is wider than summation alone; it can also support weighted average aggregation. As shown in [38], a method is proposed to enhance models' convergence speed and accuracy in a federated learning setting with non-i.i.d. data distribution by employing unbiased gradients aggregation. In this context, the server needs to aggregate $\sum_{i=1}^{n} \frac{|D_i|}{|D|} x_i$, where $|D| = \sum_{i=1}^{n} |D_i|$. To accommodate weighted average aggregation in this manner, we utilize two masks, r_i and r_i' , to encrypt the users' weighted gradients and data size. r_i' is shared among users in the same way as r_i. The encrypted data are represented as $y_i = |D_i|x_i + r_i$ and $c_i = |D_i| + r_i'$. Afterwards, these encrypted data are sent to the server. The server can decode to obtain the sum without knowing the individual values of $|D_i|x_i$ and $|D_i|$, thus achieving effective and secure weighted aggregation.

5.4. Verification Phase

In this section, we will describe the verifiability of the protocol for the aggregated results. It is mentioned in [16] that if the value of the gradients hash is sent to the server as a verification label, the gradient entries generated by SGD rules form a bell-shaped distribution around zero because the values of the model parameters (gradients) are highly partial. In addition, since all users share the same private key, this is susceptible to brute-force attacks by malicious users in collusion with the server, so that the encoded gradients can be recovered in a brute-force attack on the victim's homomorphic hash output in a relatively short time.

First, TA will send system parameters to users, including homomorphic hash parameters and a random number. The function of the random number s is to mask the gradients of the previous section twice, to prevent the server from forging the aggregation results and then taking the hash of the aggregation results as proof to deceive the user into passing the verification. User i encrypts the gradients as follows:

$$[[x_i]] = x_i + r_i + PRG(s), \tag{1}$$

where PRG is the pseudorandom generator. The user then utilizes the homomorphic hash function to generate a verification label:

$$T_i = \textbf{HHF.Hash}(x_i + r_i). \tag{2}$$

The user uploads $[[x_i]], T_i$ to the server.

Once the server receives encrypted gradients and verification labels, it performs two aggregations. The first aggregation is to calculate the aggregation results:

$$X = \sum_{u_i \in U} [[x_i]] - R = \sum_{u_i \in U} x_i + nPRG(s). \tag{3}$$

And the second aggregation is to calculate verification labels to generate the aggregation results' proof as follows:

$$T = \prod_{u_i \in U} T_i = \textbf{HHF.Hash}(\sum_{u_i \in U} (x_i + r_i)). \tag{4}$$

Then the server returns the aggregation results X and the verification proofs T, R.

After receiving the aggregation results and verification proofs from the server, the user checks the following:

$$\textbf{HHF.Hash}(X - nPRG(s) + R) = T. \tag{5}$$

If the equation is true, it represents that the server returns the correct aggregation results, and the user will use the aggregation results for the next round of training. Otherwise, the verification fails, and the user exits the protocol.

5.5. Correctness of the Scheme

In order to verify the correctness of our scheme, we prove the following theorems.

Theorem 1. *If all users and the central server honestly execute our proposed protocol, the users can obtain the correct aggregation results, and the central server can pass the verification.*

Proof. The central server aggregates all the encryption gradients as follows:

$$\sum_{u_i \in U} [[x_i]] = \sum_{u_i \in U} (x_i + r_i + PRG(s)) \quad = \sum_{u_i \in U} x_i + \sum_{u_i \in U} r_i + nPRG(s). \tag{6}$$

Next, the server calculates the aggregation results:

$$X = \sum_{u_i \in U} [[x_i]] - R = \sum_{u_i \in U} x_i + nPRG(s). \tag{7}$$

After receiving the aggregate results, the user calculates as follows:

$$X - nPRG(s) = \sum_{u_i \in U} x_i. \tag{8}$$

So, the user obtains the correct aggregation results and then performs the following hash calculation:

$$H = \textbf{HHF.Hash}(\sum_{u_i \in U} x_i + R). \tag{9}$$

If the server executes the protocol correctly, the aggregated hash is:

$$\begin{aligned} T = \prod_{u_i \in U} T_i &= \textbf{HHF.Hash}(\sum_{u_i \in U} y_i) \ = \textbf{HHF.Hash}(\sum_{u_i \in U} (x_i + r_i)) \\ &= \textbf{HHF.Hash}(\sum_{u_i \in U} x_i + \sum_{u_i \in U} r_i) = \textbf{HHF.Hash}(\sum_{u_i \in U} x_i + R). \end{aligned} \tag{10}$$

So, the server will pass the verification if Equation (9) equals Equation (10). □

Theorem 2. *Although some users may drop out of the training protocol, our proposed approach can still be effectively executed as long as the number of dropouts remains below the predefined threshold t.*

Proof. (Outline) In our proposed scheme, the impact of user dropout is limited to rounds 2 or 3. Let U_2 be the set of online users in round 2, from whom the server receives only the encryption gradients and verification labels. The server then performs an aggregation operation to eliminate $\sum_{u_i \in U_2} r_i$, and sends U_2 to the users in round 3. Users in round 3, who have saved the secret shares of U_2, send the sum of these user shares to the server. If the number of users in round 3 is greater than or equal to t (i.e., the server has received at least t shares of R), the server can reconstruct $R = \sum_{u_i \in U_2} r_i$. The verification process follows a similar approach as the aggregation process. □

Theorem 3. *The homomorphic hash function used in our approach can effectively verify the result of gradients aggregation.*

Proof. Each user computes a verification label, the hash of masked gradients $T_i = \prod_{k=1}^{m} g_k^{y_i[k]}$, where m is the gradients' size. The server aggregates the verification labels as follows:

$$\prod_{i=1}^{n} T_i = \prod_{i=1}^{n} \prod_{k=1}^{m} g_k^{y_i[k]} = \prod_{k=1}^{m} g_k^{\sum_{i=1}^{n} (x_i + r_i)[k]} = \prod_{k=1}^{m} g_k^{(X+R)[k]}. \tag{11}$$

Based on homomorphic properties, we can derive the following:

$$\prod_{i=1}^{n} T_i = \prod_{i=1}^{n} \textbf{HHF.Hash}(y_i) = \textbf{HHF.Hash}(y_1 + y_2 + ... + y_n) = \textbf{HHF.Hash}(X + R). \quad (12)$$

From this, we can conclude that the server can perform another form of aggregation on the y_i without knowing its specific value, thus achieving effective verification of the system. □

6. Security Analysis

In this section, we demonstrate the security of our scheme in terms of users' data privacy and integrity verification.

6.1. User Privacy

This section provides a demonstration that user privacy is protected. We use a similar proof method as that in [17]. We use Lemma 1 to prove that the encryption gradients of user uploads are secure.

Theorem 4. *Fix n, s, U, r_i with $|U| = n$ and $\{x_i\}_{u_i \in U}$, where $\forall u_i \in U, x_i \in Z_R^m$, there is :*

$$\{ r_i \in Z_R^m : x_i + r_i + PRG(s) (mod\ B)_{u_i \in U} \equiv I_i \in Z_R^m : I_i (mod\ B)_{u_i \in U} \}, \quad (13)$$

where "≡" means having the same distribution, we have omitted this proof as it is simple.

We use Theorem 4 to prove the privacy of the encrypted gradients uploaded by users. The encrypted gradients consist of x_i, r_i and *PRG(s)*. According to Theorem 4, the distribution of $[[x_i]]$ is the same as I_i. If an adversary possesses the sum of r_i and *PRG(s)*, he/she can obtain x_i from $[[x_i]]$ due to the random distribution of I_i. However, the adversary must acquire at least t shares to reconstruct r_i and obtain the sum of r_i and *PRG(s)*. Thus, the adversary cannot obtain any information about x_i, indicating that our scheme maintains the security of the encrypted gradients during transmission, thereby ensuring user privacy. Since the encrypted gradients are not highly partial for the hash value uploaded by users, the encrypted gradients are hashed and then uploaded to the server. So, a malicious server cannot infer users' private data according to the uploaded results, and it will not lead to the model recovery attacks like the one mentioned in [16].

6.2. Integrity Verification

Theorem 5. *According to the collision resistance and one way of **HHF**, a server cannot forge the aggregation results to pass the verification. The server returns the aggregation results X and the verification proofs T, R. Assume $V = X + R$; the result returned by the server is represented by V, T. We assume that the server is the adversary. There are only two ways for the server to falsify results, and we will show that falsifying proofs in either of these ways will not pass verification.*

(1) The server tries to forge V or T to pass the verification: In this case, the server forges V or T in round 3. We denote the forged results as V_f, T_f and the correct ones as V, T. Assume that the server only forges V to pass the verification, so $V \neq V_f$ and $T = T_f$. Let H denote the correct hash value computed by the user and H_f represent the falsified hash value. We can obtain that from the third section:

$$H = T, H_f = T_f. \quad (14)$$

Since $T = T_f$, we can get $H = H_f$ according to Equation (14). However,

$$\begin{aligned} H &= \textbf{HHF.Hash}(V - nPRG(s)) \\ H_f &= \textbf{HHF.Hash}(V_f - nPRG(s)). \end{aligned} \quad (15)$$

According to the collision resistance of **HHF**, we can conclude that $H \neq H_f$, since $V \neq V_f$. This contradicts our previous assumption. The same argument holds true if the server forges T instead. Therefore, the server cannot pass the verification by forging V or T.

(2) The server tries to forge V and T to pass the verification: In this case, since the server can pass the verification, the following equation that users calculate holds:

$$H_f = \textbf{HHF.Hash}(V_f - nPRG(s)) = T_f. \tag{16}$$

The adversary did not know s and $V \neq V_f$ because the adversary did not execute the aggregation protocol correctly. So,

$$T_f = \textbf{HHF.Hash}(s'), \tag{17}$$

where $s' \in Z_q^*$ is a random vector guessed by the adversary. According to the one way of **HHF**, the probability of making Equation (16) valid is negligible. Therefore, the probability that the server wants to falsify the proofs to pass the verification is also negligible.

7. Model Analysis

7.1. Experimental Setup

We executed our experiment on a workstation with Java1.8 running Windows 11, equipped with an IntelCore i7-11700K 3.6GHz CPU and 16.0GB of RAM. We used elliptic curve Diffie-Hellman, standard Shamir's t-out-of-N secret sharing and advanced encryption standard galois to implement **KA**, **SS** and **AE**, respectively.

7.2. Comparison with the Other Two Experiments Verifl [15] and Versa [16]

7.2.1. Computation Overhead

In our scheme and the other two schemes, we compared the computational costs at different stages, as shown in Table 2. We omitted the comparison of the server overhead because the server is typically a powerful entity with negligible aggregation costs. Additionally, we omitted the consideration of round 0 as it does not incur any computation overhead. The underlined figures represent the verification costs. We fix the gradients' size to 10,000.

Table 2. Comparison of computation overhead with Versa and Verifl.

		Ours		**Versa**		**Verifl**	
		$n = 500$	$n = 1000$	$n = 500$	$n = 1000$	$n = 500$	$n = 1000$
Aggregation phase	1	2038 ms	4103 ms	2262 ms	4601 ms	781 + 7034 ms	1572 + 14,252 ms
	2	204 + 37,900 ms	210 + 37,903 ms	6123 + 6385 ms	6098 + 6465 ms	6236 ms	6248 ms
	3	2030 ms	4010 ms	2130 ms	4012 ms	2725 ms	5821 ms
Verification phase	4	37,913 ms	37,962 ms	120 ms	125 ms	2330 ms	4805 ms

The table shows that our aggregation cost is slightly lower than the other two approaches, while the verification cost is significantly higher. Our approach's lower aggregation cost is attributed to the fact that we only use a single mask for aggregation. In contrast, the other two approaches employ double-mask encrypted gradients, requiring each user to compute the shared value ciphertext of two masks. The higher verification cost is due to adopting a more secure hash value to counter model recovery attacks from the adversary. In Versa, shared vector computation is used for verification labels. In VeriFL, each user employs linear homomorphic hashing and commitment for generating verification labels, which can lead to model recovery attacks, posing a significant privacy risk to users.

In the federated learning of privacy protection, user privacy, communication costs and the verifiability of the aggregated results are key issues during training, considering that modern edge mobile devices are typically equipped with high-performance processors but limited network bandwidth. Research [7,39] optimize the performance and efficiency

of federated learning systems by increasing local computations and minimizing communication volumes. Furthermore, when it comes to transmitting sensitive data, a more cautious approach is required. It is essential to leverage local computational resources to minimize the transmission of private data. As seen in Section 7.2.2, the communication cost of our verification transmission is minimal, indicating that our approach is meant to reduce communication costs by increasing local computation overhead on edge mobile devices.

7.2.2. Communication Overhead

We also compared the other two schemes in terms of the communication overhead for aggregation and verification.

Figure 3 depicts the aggregated communication cost between each user and the server. It demonstrates that our proposed approach, and the other two methods, increase linearly with the gradients' size and the number of users in outgoing communication costs. Moreover, our protocol exhibits lower costs than the other two approaches. This discrepancy arises from using a single masking technique for aggregation in our scheme, whereas the other two employ double masking techniques. In the double-mask scheme, users must share two mask values to ensure the security of the aggregation.

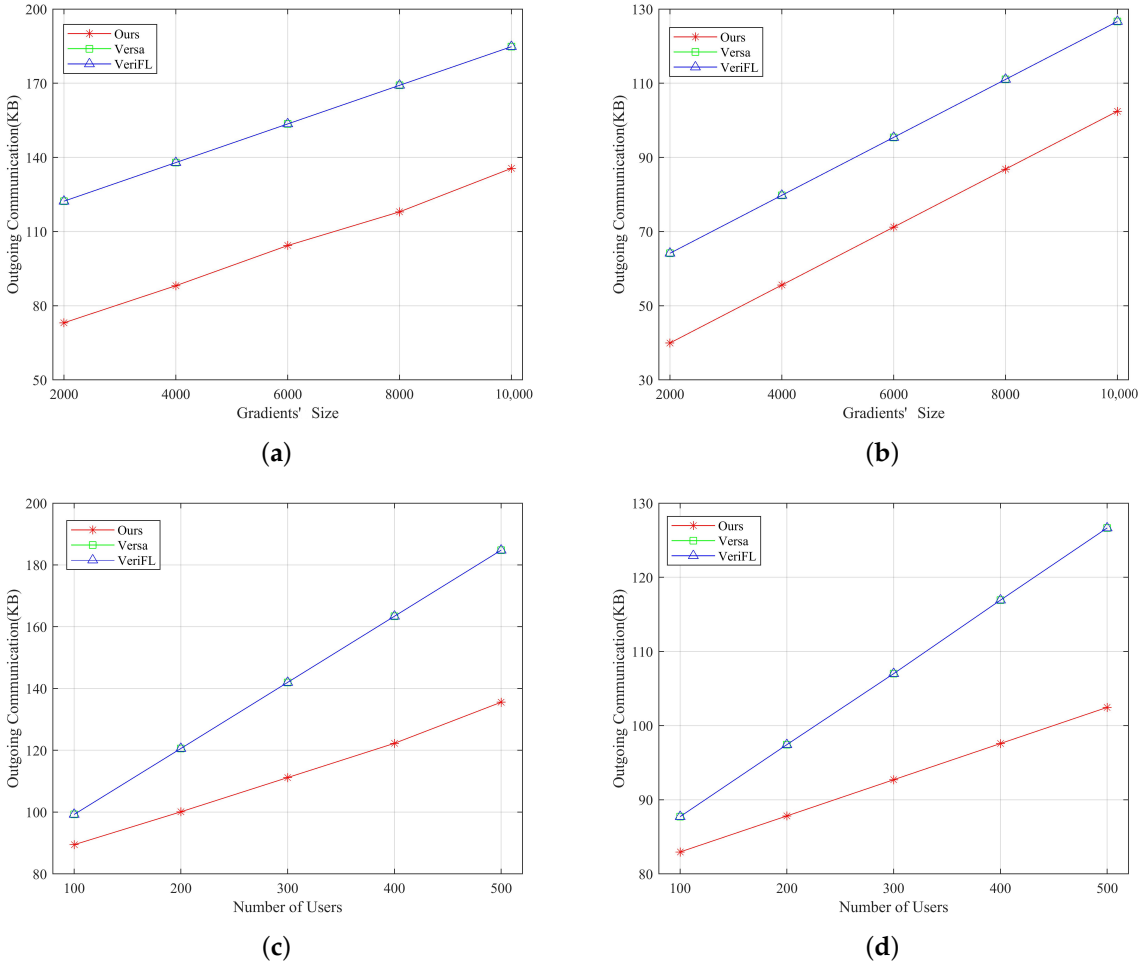

Figure 3. Comparison of outgoing communication overhead of our scheme with Versa and Verifl for aggregation. (**a**) Outgoing communication overhead of the server as the gradients' size increases for aggregation. (**b**) Outgoing communication overhead per user as the gradients' size increases for aggregation. (**c**) Outgoing communication overhead of the server compared to the different number of users for aggregation. (**d**) Outgoing communication overhead per user compared to the different number of users for aggregation.

Figure 4 illustrates the outgoing communication costs for verification between each user and the server. It shows that our proposed verification approach exhibits the lowest outgoing communication costs for both the user and the server, independent of the gradients's size or the number of users. This advantage stems from transmitting only a single hash value for verification. However, in VeriFL, these two metrics are independent of the gradients' size and increase linearly with the number of users. This is due to the necessity of users sharing their hash and commitment values. In contrast, these two metrics are independent of the number of users in Versa and increase linearly with the gradients' size. In their approach, users must send a verification vector of the same dimensions as the gradients. Therefore, our verification method is particularly suitable for scenarios with multiple participating users and high-dimensional gradients.

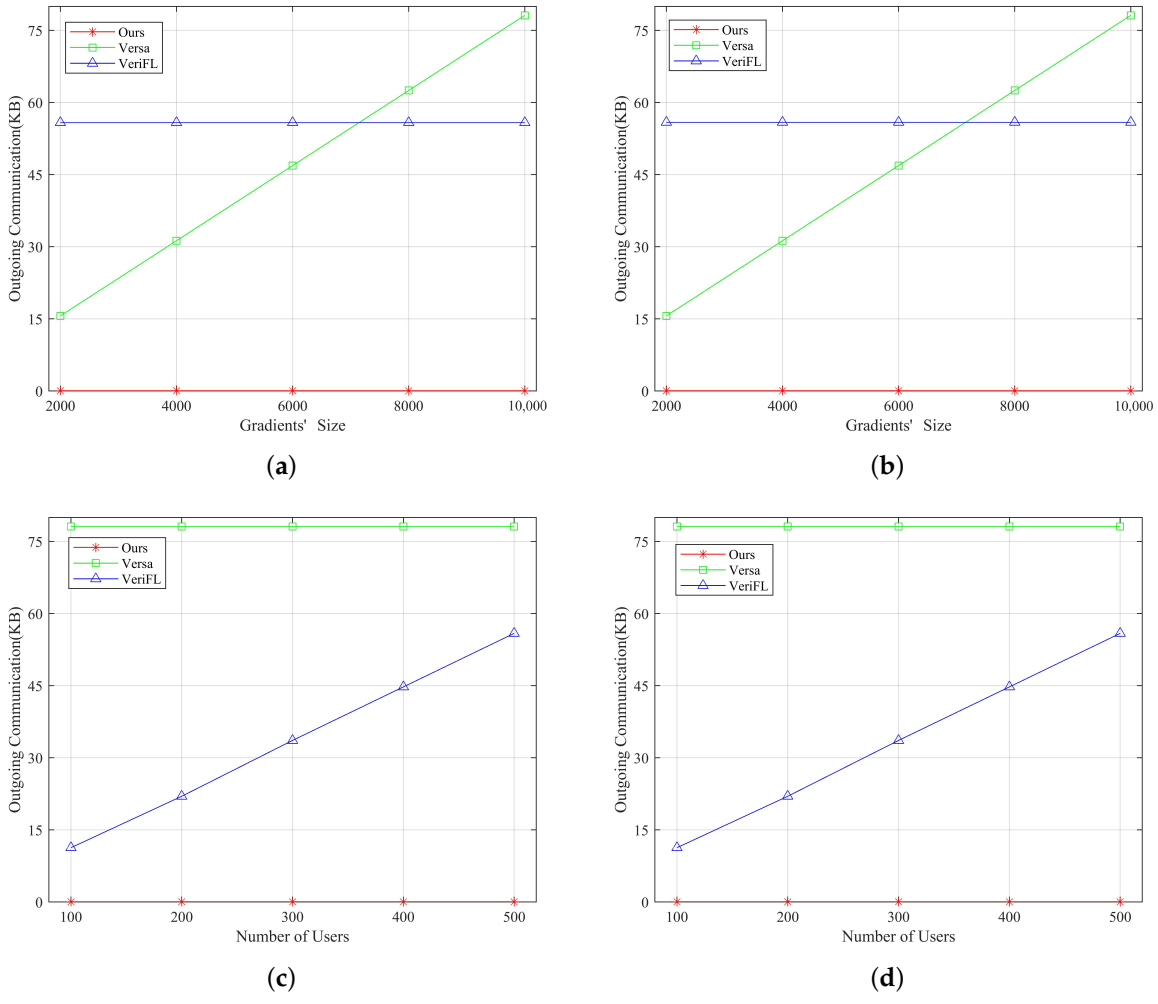

Figure 4. Comparison of outgoing communication overhead of our scheme with Versa and Verifl for verification. (**a**) Outgoing communication overhead of the server as the gradients' size increases for verification. (**b**) Outgoing communication overhead per user as the gradients' size increases for verification. (**c**) Outgoing communication overhead of the server compared to the different number of users for verification. (**d**) Outgoing communication overhead per user compared to the different number of users for verification.

7.3. Other Experiments of Our Scheme

This section analyzes our scheme's computational and communication costs under different user dropout rates. In our experiment, we set the number of users $n = 500$, and the gradients' size $m = 10,000$.

7.3.1. Computation Overhead

From Figure 5a,c, it can be seen that the running time of the server decreases as the user dropout rate increases. The main reason is that the server only needs to aggregate gradients and tags and recover shares of online users. As shown in Figure 5b, the dropout rate does not affect the computational cost, as most of the time is spent on computing hashes, while the decryption time of ciphertext for online users can be ignored. Figure 5d indicates that the user's computational cost of dropout is lower than that without dropout. This is because, in round 3, the user decrypts the online users' ciphertext. Based on the experimental results, our scheme exhibits stronger user dropout robustness, as evidenced by a decrease in its running time with an increase in the dropout rate.

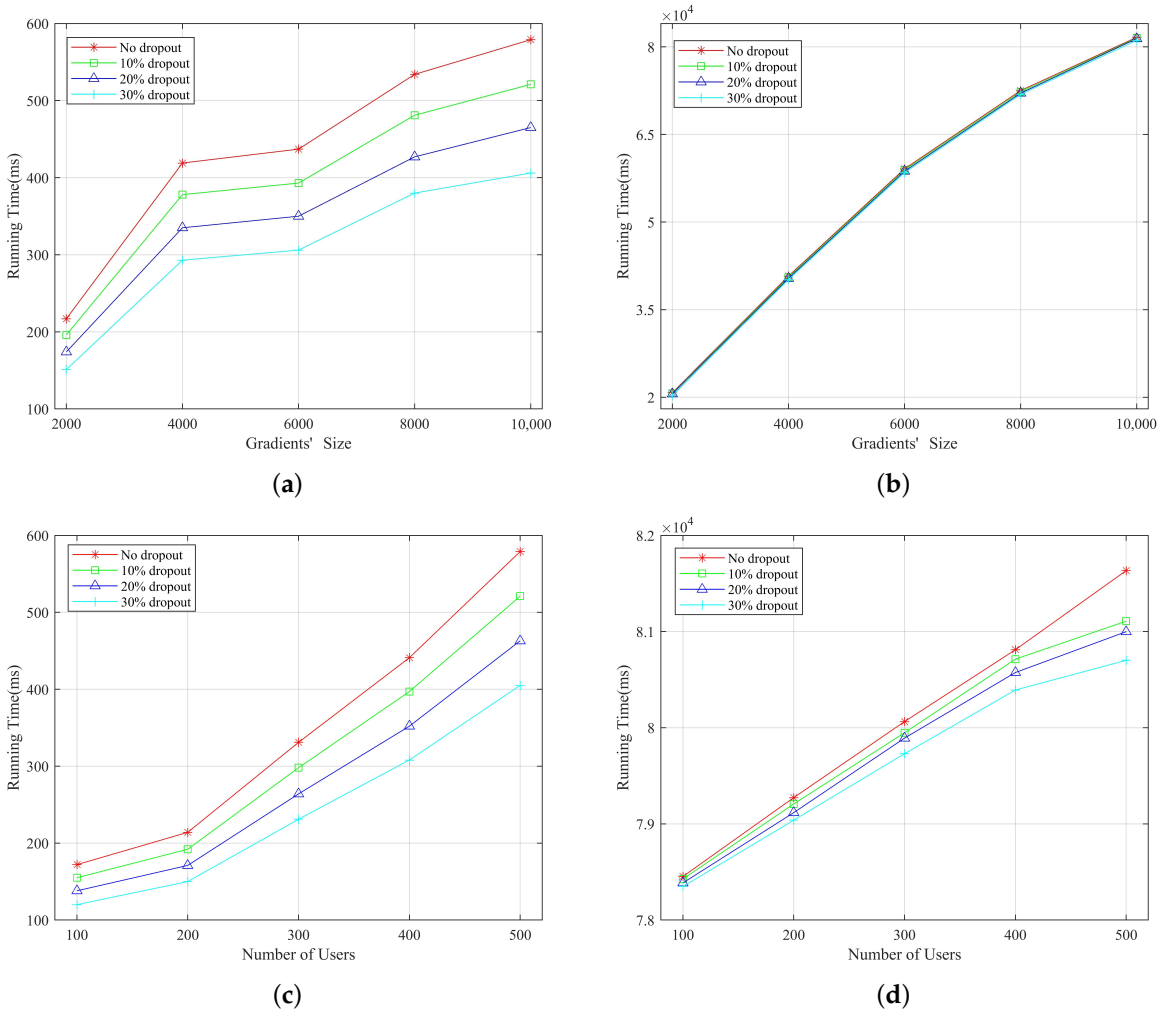

Figure 5. Thetotal computational overhead of the server and each user (**a**) computational overhead of the server versus different gradients' size and dropout rates (**b**) computational overhead of each user versus different gradients' size and dropout rates (**c**) computational overhead of the server versus different user numbers and dropout rates (**d**) computational overhead of each user versus different user numbers and dropout rates.

7.3.2. Communication Overhead

Figure 6 shows the comparison of communication overhead and dropout rate between the server and each user. Specifically, Figure 6a and Figure 6c respectively demonstrate that the server's communication overhead increases linearly with the number of users and gradients' size and is independent of the dropout rates since the data sent by the server to each user remain constant regardless of the dropout rate. Additionally, Figure 6b,d show that the communication overhead per user under different dropout rates is uniform. Table 3 presents the outgoing communication overhead between the server and per user during different phases of our protocol. The underlined figures highlight the additional communication overhead associated with verification. Based on the table, we infer that the additional communication overhead caused by our verification process is minimal. This further confirms that our scheme is suitable for limited communication bandwidth settings.

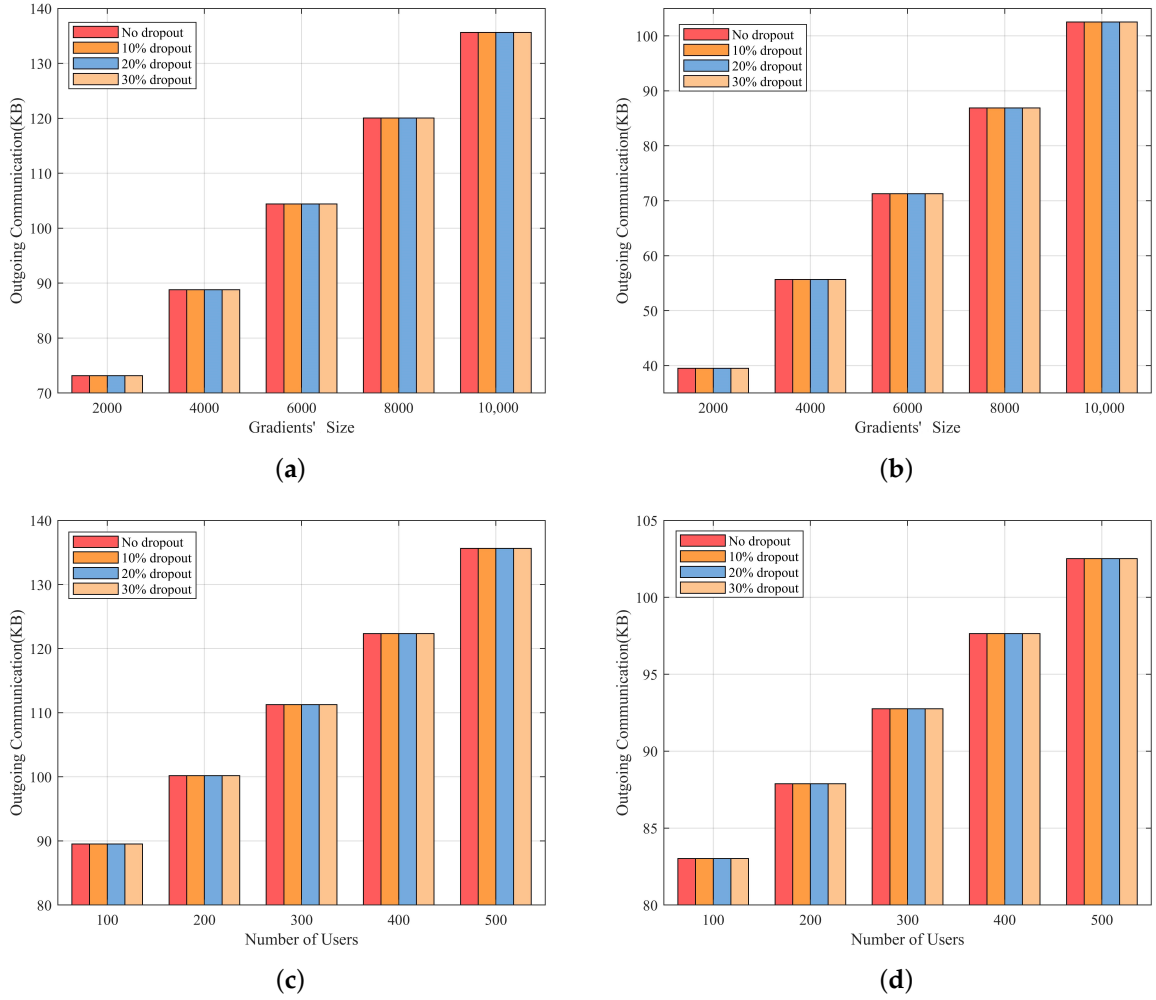

Figure 6. Total communication overhead between the server and each user (**a**) The communication overhead of the server versus different gradients' size and dropout rates (**b**) the communication overhead per user versus different gradients' size and dropout rates (**c**) The communication overhead of the server versus a different number of users and dropout rates (**d**) the communication overhead per user versus a different number of users and dropout rates.

Table 3. Outgoing communication overhead per user and server at different phases.

Num. User	Dropout	Entity	Aggregation Phase				Verification Phase
			0	**1**	**2**	**3**	**4**
500	0.00%	user	0.06 (KB)	24.23 (KB)	78.12 + 0.06 (KB)	0.03 (KB)	0 (KB)
		server	32.23 (KB)	24.23 (KB)	0.98 (KB)	78.12 + 0.07 (KB)	0 (KB)
	10.00%	user	0.06 (KB)	24.23 (KB)	78.12 + 0.06 (KB)	0.03 (KB)	0 (KB)
		server	32.23 (KB)	24.23 (KB)	0.88 (KB)	78.12 + 0.07 (KB)	0 (KB)
	30.00%	user	0.06 (KB)	24.23 (KB)	78.12 + 0.06 (KB)	0.03 (KB)	0 (KB)
		server	32.23 (KB)	24.23 (KB)	0.68 (KB)	78.12 + 0.07 (KB)	0 (KB)

8. Conclusions

This paper proposed a secure aggregation method that utilizes a single mask to encrypt gradients, providing a high level of privacy protection for users' local gradients and allowing them to dropout. In addition, we devised a novel approach to verify the integrity of the aggregation results by hashing the masked value, capable of resisting the adversary's attacks on the hash value in the federated learning framework. The experimental results demonstrate the efficiency of our approach in terms of communication costs for aggregation and verification. Therefore, our approach is effective in scenarios with limited bandwidth resources, solid computational capabilities, and frequent dropout of mobile devices.

However, our method also has some limitations. Firstly, the computational overhead of our scheme validation may be higher, which may not be suitable for devices with lower computing power, such as IoT devices and edge computing nodes. Secondly, our scheme requires five rounds of interaction to complete one iteration of training, leading to significant communication latency that hampers model training. Thirdly, in the event of incorrect results returned by the server, users cannot correct the aggregated results through available means. Therefore, it is necessary to further study the methods of reducing communication times and computing costs to achieve verifiable aggregation in federated learning training protocols.

Author Contributions: Conceptualization, X.S. and Y.L.; Methodology, K.P.; Writing—original draft, K.P.; Writing—review & editing, X.S.; Supervision, B.W.; Funding acquisition, L.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China (2022YFC3303200), the National Natural Science Foundation of China (Grant Nos. 62272362, U19B2021), the Open Research Fund of Key Laboratory of Cryptography of Zhejiang Province (ZCL21016), the Fundamental Research Funds for the Central Universities (XJS220122), the Teaching Reform Project of Guangdong Province (Grant Nos.GDJX2020009), and the Information Security Teaching Reform Project of Wuyi University (Grant Nos.JX2020052).

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare that they have no known competing financial interest or personal relationship that could have appeared to influence the work reported in this paper.

Abbreviations

FL	federated learning
SS	secret sharing
HHF	homomorphic hash function
KA	key agreement
AE	authenticated encryption
TA	trusted third party
DP	Differential privacy
HE	homomorphic encryption
SMC	secure multi-party computation
PPML	privacy-preserving machine learning
PPDL	privacy-preserving deep learning

References

1. Li, L.; Fan, Y.; Tse, M.; Lin, K.-Y. A review of applications in federated learning. *Comput. Ind. Eng.* **2020**, *149*, 106854. [CrossRef]
2. Konečnỳ, J.; McMahan, H.B.; Yu, F.X.; Richtárik, P.; Suresh, A.T.; Bacon, A.D. Federated learning: Strategies for improving communication efficiency. *arXiv* **2016**, arXiv:1610.05492.
3. Połap, D.; Srivastava, G.; Yu, K. Agent architecture of an intelligent medical system based on federated learning and blockchain technology. *J. Inf. Secur. Appl.* **2021**, *58*, 102748. [CrossRef]
4. Xing, E.P.; Ho, Q.; Xie, P.; Wei, D. Strategies and principles of distributed machine learning on big data. *Engineering* **2016**, *2*, 179–195. [CrossRef]
5. Antwi-Boasiako, E.; Zhou, S.; Liao, Y.; Liu, Q.; Wang, Y.; Owusu-Agyemang, K. Privacy preservation in distributed deep learning: A survey on distributed deep learning, privacy preservation techniques used and interesting research directions. *J. Inf. Secur. Appl.* **2021**, *61*, 102949. [CrossRef]
6. Bonawitz, K.; Eichner, H.; Grieskamp, W.; Huba, D.; Ingerman, A.; Ivanov, V.; Kiddon, C.; Konečnỳ, J.; Mazzocchi, S.; McMahan, B.; et al. Towards federated learning at scale: System design. *Proc. Mach. Learn. Syst.* **2019**, *1*, 374–388.
7. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A.Y. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*; PMLR: London, UK, 2017; pp. 1273–1282.
8. McMahan, H.B.; Ramage, D.; Talwar, K.; Zhang, L. Learning differentially private recurrent language models. *arXiv* **2017**, arXiv:1710.06963.
9. Aono, Y.; Hayashi, T.; Wang, L.; Moriai, S. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Trans. Inf. Forensics Secur.* **2017**, *13*, 1333–1345.
10. Wang, Z.; Song, M.; Zhang, Z.; Song, Y.; Wang, Q.; Qi, H. Beyond inferring class representatives: User-level privacy leakage from federated learning. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 2512–2520.
11. Zhao, Q.; Zhao, C.; Cui, S.; Jing, S.; Chen, Z. Privatedl: Privacy-preserving collaborative deep learning against leakage from gradient sharing. *Int. J. Intell. Syst.* **2020**, *35*, 1262–1279. [CrossRef]
12. Ghodsi, Z.; Gu, T.; Garg, S. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 4672–4681.
13. Fu, A.; Zhu, Y.; Yang, G.; Yu, S.; Yu, Y. Secure outsourcing algorithms of modular exponentiations with optimal checkability based on a single untrusted cloud server. *Clust. Comput.* **2018**, *21*, 1933–1947. [CrossRef]
14. Xu, G.; Li, H.; Liu, S.; Yang, K.; Lin, X. Verifynet: Secure and verifiable federated learning. *IEEE Trans. Inf. Forensics Secur.* **2019**, *15*, 911–926. [CrossRef]
15. Guo, X.; Liu, Z.; Li, J.; Gao, J.; Hou, B.; Dong, C.; Baker, T. V eri fl: Communication-efficient and fast verifiable aggregation for federated learning. *IEEE Trans. Inf. Forensics Secur.* **2020**, *16*, 1736–1751. [CrossRef]
16. Hahn, C.; Kim, H.; Kim, M.; Hur, J. Versa: Verifiable secure aggregation for cross-device federated learning. *IEEE Trans. Dependable Secur. Comput.* **2023**, *20*, 36–52. [CrossRef]
17. Song, J.; Wang, W.; Gadekallu, T.R.; Cao, J.; Liu, Y. Eppda: An efficient privacy-preserving data aggregation federated learning scheme. *IEEE Trans. Netw. Sci. Eng.* 2022, *early access*. [CrossRef]
18. 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, Vienna, Austria, 24–28 October 2016; pp. 308–318.
19. Zhou, J.; Wu, N.; Wang, Y.; Gu, S.; Cao, Z.; Dong, X.; Choo, K.-K.R. A differentially private federated learning model against poisoning attacks in edge computing. *IEEE Trans. Dependable Secur.* **2023**, *20*, 1941–1958. [CrossRef]
20. Xu, M.; Song, C.; Tian, Y.; Agrawal, N.; Granqvist, F.; van Dalen, R.; Zhang, X.; Argueta, A.; Han, S.; Deng, Y.; et al. Training large-vocabulary neural language models by private federated learning for resource-constrained devices. In Proceedings of the ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Rhodes Island, Greece, 4–9 June 2023; pp. 1–5.
21. Park, J.; Lim, H. Privacy-preserving federated learning using homomorphic encryption. *Appl. Sci.* **2022**, *12*, 734. [CrossRef]

22. Du, W.; Li, M.; Wu, L.; Han, Y.; Zhou, T.; Yang, X. A efficient and robust privacy-preserving framework for cross-device federated learning. *Complex Intell. Syst.* **2023**. [CrossRef]
23. 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, Dallas, TX, USA, 30 October–3 November 2017; pp. 1175–1191.
24. Lu, S.; Li, R.; Liu, W.; Guan, C.; Yang, X. Top-k sparsification with secure aggregation for privacy-preserving federated learning. *Comput. Secur.* **2023**, *124*, 102993. [CrossRef]
25. So, J.; Güler, B.; Avestimehr, A.S. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE J. Sel. Areas Inf. Theory* **2021**, *2*, 479–489. [CrossRef]
26. Mohassel, P.; Zhang, Y. Secureml: A system for scalable privacy-preserving machine learning. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 19–38.
27. So, J.; He, C.; Yang, C.-S.; Li, S.; Yu, Q.; Ali, R.E.; Guler, B.; Avestimehr, S. Lightsecagg: A lightweight and versatile design for secure aggregation in federated learning. *Proc. Mach. Learn. Syst.* **2022**, *4*, 694–720.
28. Schlegel, R.; Kumar, S.; Rosnes, E.; Amat, A.G.I. Codedpaddedfl and codedsecagg: Straggler mitigation and secure aggregation in federated learning. *IEEE Trans. Commun.* **2023**, *71*, 2013–2027. [CrossRef]
29. Zhao, Y.; Sun, H. Information theoretic secure aggregation with user dropouts. *IEEE Trans. Inf. Theory* **2022**, *68*, 7471–7484. [CrossRef]
30. Jahani-Nezhad, T.; Maddah-Ali, M.A.; Li, S.; Caire, G. Swiftagg+: Achieving asymptotically optimal communication loads in secure aggregation for federated learning. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 977–989. [CrossRef]
31. Zhang, X.; Fu, A.; Wang, H.; Zhou, C.; Chen, Z. A privacy-preserving and verifiable federated learning scheme. In Proceedings of the ICC 2020–2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–6.
32. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613. [CrossRef]
33. Benaloh J.C. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Conference on the Theory and Application of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 1986; pp. 251–260.
34. Krohn, M.N.; Freedman, M.J.; Mazieres, D. On-the-fly verification of rateless erasure codes for efficient content distribution. In Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 9–12 May 2004; pp. 226–240.
35. Diffie, W.; Hellman, M. New directions in cryptography. *IEEE Trans. Inf. Theory* **1976**, *22*, 644–654. [CrossRef]
36. Rogaway, P. Authenticated-encryption with associated-data. In Proceedings of the 9th ACM Conference on Computer and Communications Security, Kyoto, Japan, 4–6 June 2014; pp. 98–107.
37. Gordon, S.D.; Katz, J.; Liu, F.-H.; Shi, E.; Zhou, H.-S. Multi-client verifiable computation with stronger security guarantees. In Proceedings of the 2th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, 23–25 March 2015; pp. 144–168.
38. Yao, X.; Huang, T.; Zhang, R.-X.; Li, R.; Sun, L. Federated learning with unbiased gradient aggregation and controllable meta updating. *arXiv* **2019**, arXiv:1910.08234.
39. Vepakomma, P.; Gupta, O.; Swedish, T.; Raskar, R. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv* **2018**, arXiv:1812.00564.