electronics

MDPI

Article

An Efficient Multi-Party Secure Aggregation Method Based on Multi-Homomorphic Attributes

Qi Gao [1], Yi Sun [1,*], Xingyuan Chen [1,2,*], Fan Yang [1] and Youhe Wang [1]

[1] School of Cryptographic Engineering, Information Engineering University, Zhenzhou 450001, China; qig_57@163.com (Q.G.)
[2] State Key Laboratory of Cryptology, Beijing 100094, China
* Correspondence: 11112072@bjtu.edu.cn (Y.S.); chxy302@vip.sina.com (X.C.)

Abstract: The federated learning on large-scale mobile terminals and Internet of Things (IoT) devices faces the issues of privacy leakage, resource limitation, and frequent user dropouts. This paper proposes an efficient secure aggregation method based on multi-homomorphic attributes to realize the privacy-preserving aggregation of local models while ensuring low overhead and tolerating user dropouts. First, based on EC-ElGamal, the homomorphic pseudorandom generator, and the Chinese remainder theorem, an efficient random mask secure aggregation method is proposed, which can efficiently aggregate random masks and protect the privacy of the masks while introducing secret sharing to achieve tolerance of user dropout. Then, an efficient federated learning secure aggregation method is proposed, which guarantees that the computation and communication overheads of users are only $O(L)$; also, the method only performs two rounds of communication to complete the aggregation and allows user dropout, and the aggregation time does not increase with the dropout rate, so it is suitable for resource-limited devices. Finally, the correctness, security, and performance of the proposed method are analyzed and evaluated. The experimental results indicate that the aggregation time of the proposed method is linearly related to the number of users and the model size, and it decreases as the number of dropped out users increases. Compared to other schemes, the proposed method significantly improves the aggregation efficiency and has stronger dropout tolerance, and it improves the efficiency by about 24 times when the number of users is 500 and the dropout rate is 30%.

Keywords: secure aggregation; privacy protection; federated learning; tolerate dropouts; homomorphic

check for updates

Citation: Gao, Q.; Sun, Y.; Chen, X.; Yang, F.; Wang, Y. An Efficient Multi-Party Secure Aggregation Method Based on Multi-Homomorphic Attributes. *Electronics* **2024**, *13*, 671. https://doi.org/10.3390/electronics13040671

Academic Editor: Zbigniew Kotulski

Received: 8 December 2023
Revised: 6 January 2024
Accepted: 15 January 2024
Published: 6 February 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Federated learning (FL) [1] is a distributed machine learning framework proposed by Google, which enables multiple distributed clients to train models using local private data, and then the global model is established by aggregating the local models from the clients through a central server to achieve distributed collaborative training. FL, which ensures that user data are kept local and allows for the training of high-quality models using user data, has been extensively investigated in various fields [2,3] in recent years. With the development of the mobile Internet of Things (IoT), an increasing amount of devices access the network, such as mobile terminals, IoT devices [4], etc., which are huge in scale and contain a large amount of valuable data. This makes it possible for enterprises to train high-quality models by utilizing these data for FL to provide a better user experience and enhance the competitiveness of the industry [5].

Compared with traditional model training approaches that collect raw client data, FL can significantly reduce the risk of user data privacy leakage by sharing models. However, existing studies have shown that sensitive information in the user's local dataset can still be obtained by implementing privacy attacks on the local model, such as membership inference attacks [6,7], attribute inference attacks [8], etc. To solve the above problems, many

studies [9–12] have proposed privacy-preserving mechanisms based on techniques such as secure multi-party computation (MPC), homomorphic encryption (HE), and differential privacy (DP) [13] to protect local model parameters of FL users. Specifically, DP [14,15] involves the user perturbing the local model by carefully adding constructed artificial noise to the model parameters before uploading the local model to the aggregation server, to prevent the server from accessing the original local model parameters. Although DP-based protection is efficient, it requires an effective trade-off between the level of privacy protection and model utility: more noise leads to a better protection effect but significantly affects the accuracy of the global model. Based on the secret sharing technique in MPC, works from the literature [16–18] allowed users to share model parameters with multiple users or servers, and a single or a small number of users and servers cannot reconstruct the user's local model. In contrast, other studies found in the literature [10,19,20] designed HE-based schemes, where the user encrypts the model parameters to protect the model, then sends the ciphertext of the parameters to the server for aggregation, and finally the server returns the aggregated results to the user for decryption; in this process, the homomorphic encryption property ensures the correctness of the aggregated results. HE and MPC basically do not affect the quality of the global model, but they are not suitable for large-scale FL where the encryption of model parameters or secret sharing will result in huge additional computational and communication overheads for users. In particular, in cross-device FL, the participants are usually numerous mobile terminals with limited computation and communication resources, IoT devices [12,21], etc., and the device performance may not meet the training requirements. Some studies [22,23] use a three-tier structure of IoT device-fog node-cloud, where the IoT device adds noise to the data and sends them to the fog node, which trains the local model, and noise addition to the raw data reduces the quality of the data, which in turn affects the local model. In addition, the use of HE or DP to achieve model privacy aggregation still results in either efficiency or model performance decrease. Additionally, the use of HE may require additional security assumptions, such as no collusion between servers and users.

The paper [24] proposed SecAgg, a more practical security aggregation scheme that protects the model by masking it with a random mask. Users u and v share a seed $s_{u,v}$ with each other, and then a pairwise masking mechanism enables the masks to cancel each other after aggregation to obtain the global model. However, to negotiate pairwise masks, users need to communicate with the server in multiple rounds and perform a large number of key agreements, causing the aggregation overhead to have a quadratic complexity relationship to the number of users. Additionally, to remove the effect of dropped out users, the aggregation time increases significantly with the number of dropped out users, and when d users drop out, the server needs to add $(n-d)d$ key reconfigurations to ensure correct aggregation, so the scheme may not be applicable to the FL training scenarios with large-scale user participation and frequent dropouts. Drawing on this, some studies [25–29] proposed more efficient secure aggregation schemes, but they either lose some privacy guarantees or dropout tolerance, or require additional security assumptions, etc.

To address the above problems, in order to achieve local model privacy-preserving aggregation while avoiding the efficiency or model performance degradation caused by HE, DP, and secret sharing, this paper proposes an efficient privacy aggregation method for FL models based on multi-homomorphic attributes. Our method can guarantee low overhead and high accuracy without introducing additional assumptions. Specifically, the proposed method adopts independent masks to protect local models, thus avoiding the drawbacks brought by encrypting model parameters, sharing them secretly, or adding noise, and it also avoids the bottleneck brought by using pairwise masks in SecAgg. Then, a multi-key privacy aggregation algorithm is proposed based on EC-ElGamal, where the user encrypts the inputs with a public key to enable the server to decrypt the sum of the user's inputs using its own private key without knowing the individual inputs. Based on this algorithm, model privacy-preserving aggregation can be achieved without losing model quality while tolerating the collusion between $n-2$ users and the server. Additionally, to reduce the

user overhead and improve the aggregation efficiency, an efficient random mask privacy aggregation method is proposed based on the homomorphic pseudorandom generator (HPRG) and the Chinese remainder theorem (CRT), which can avoid the computation and communication overhead caused by users' encryption of high-dimensional masks and thus improve the overall aggregation efficiency. Finally, the tolerance of user dropouts is achieved by integrating secret sharing.

The main contributions of this paper are as follows:

- An FL secure aggregation method is proposed, which protects model privacy through independent random masks while ensuring efficiency and accuracy, with client computation and communication overheads of only $O(L)$ and without relying on trusted third parties.
- Without considering user dropouts, the proposed method can complete aggregation in only two rounds of communication and can tolerate $n-2$ users colluding with the server. Also, the method supports user dropouts, and the aggregation time decreases as the number of dropped out users increases.
- The correctness and security of the proposed method are analyzed, and its performance is evaluated through experimentation and comparison, and the results indicate that the proposed method is efficient and has stronger dropout tolerance.

The rest of the paper is organized as follows. Section 2 reviews the related work; Section 3 introduces the preliminaries; Section 4 details the scheme proposed in this paper; Section 5 analyzes the correctness and security of the scheme; Section 6 evaluates the performance of the scheme; and Section 7 concludes this paper.

2. Related Work

This paper focuses on the issue of secure aggregation in FL, i.e., securing a user's local model and preventing potential adversaries from obtaining sensitive information about the user's local dataset through privacy attacks on the local model. Also, this paper considers the issue of possible frequent dropouts of resource-limited clients in cross-device FL.

Several studies [10,19,20] implemented secure aggregation via HE techniques, such as Paillier. Homomorphic encryption allows the server to perform aggregation over ciphertexts. Although it can enhance the privacy of FL, it requires the sharing of private keys among all users, which is not resistant to user–server collusion. Some schemes [30,31] use multi-key homomorphic encryption to solve the key management problem, but encrypting model parameters via homomorphic encryption usually incurs a huge computational overhead, and parameter ciphertext transmission also increases the communication overhead of the client, so this scheme is generally not applicable in cross-device FL. Some other studies [16,18,32] achieve secure aggregation by using the secret share technique in MCP, where users share their local model parameters secretly to multiple servers or users, and based on the property of the secret share technique, a single or a small number of users cannot reconstruct the user's local model, and finally, by exploiting the homomorphism of the secret share, the users or servers work together to complete the aggregation. However, the secret sharing of model parameters and transmission of secret shares will lead to problems such as large computational and communication overheads. Moreover, the security of this scheme relies on the number of non-colluding participants, and this can only be guaranteed if there is no collusion among the participants or if most of them are trustworthy, while the reliability of FL relies on the reliability of the servers and the users, who are usually required to always be online. DP reduces the risk of privacy leakage mainly by adding artificial noise to the model parameters [14,33,34], but this requires a trade-off between the level of privacy protection and the quality of the model; a higher level of privacy protection requires adding more noise, which inevitably decreases the quality of the model.

A study [24] proposed the first mask-based secure aggregation scheme in FL called SecAgg, which masks the model with random masks. To improve security and tolerate user dropouts at any time, SecAgg implements a pairwise masking mechanism through

key agreement and secret sharing. However, this approach requires multiple rounds of communication between users and servers in a single aggregation. Also, negotiating pairwise masks requires secondary communication and computational complexity that increases with the number of users. When removing masks, the computational time needed to reconstruct these masks increases substantially with the number of dropouts, which makes SecAgg not suitable for applications in FL where a large number of users participate in training and users drop out frequently.

To achieve more efficient and secure aggregation, several studies have proposed that FL users communicate with only a subset of users rather than with all users. A study [25] aggregated n FL users by dividing them into $n/\log n$ groups and following a multi-group round-robin structure; each FL user in a group communicates with the user in the next group, which requires additional rounds of communication between the groups. The studies of [35,36] also use a group aggregation structure, where users in the same group perform intra-group aggregation through model secret sharing, and then perform inter-group aggregation with the next group. This group structure results in the need for users to increase the number of additional communications and also reduces user dropout tolerance. The study [37] reduced per-user overhead by using fast Fourier transform multi-secret sharing. However, these two approaches have poor privacy guarantees and user dropout tolerance. The works [26,27] reduced the aggregation overhead by replacing fully connected graph communication with non-fully connected graph communication so that each user only needs to perform key agreement and secret sharing with some users, compared to SecAgg, which improves the communication efficiency but reduces a certain privacy guarantee. Meanwhile, due to the retention of the double-masking mechanism, once a large number of users drop out, the server still needs to consume a large amount of time to eliminate the impact caused by the dropped out users. The studies [28,29,38] improved the efficiency of key agreements by replacing the mask negotiation algorithm. Particularly, the authors of [29] improved the efficiency of key agreements by introducing two key service providers and by designing an interaction-free key negotiation protocol. The authors of [28] designed a lightweight masking approach with additive homomorphism to eliminate the key agreement part while guaranteeing that the mask can be eliminated after aggregation. While both approaches rely on the security assumption of non-collusion, the work [38] utilized a key derivation function to generate pairwise masks for the masking model. For the user dropout training problem, the researchers eliminated the effect of the dropped out user at once by having the surviving user submit the mask of the dropped out user, which requires the surviving user to submit the entire random mask of the dropped out user to help the server complete the aggregation, thus increasing the communication overhead of the user. Meanwhile, this approach relies on the assumption that no more users drop out from training during the mask removal process, which cannot be guaranteed in cross-device FL learning. The research papers [39,40] adopted the idea of a one-time reconfiguration of aggregated masks by introducing a third-party trusted server to either aggregate the masks or distribute them, which avoids the additional computational and communication overheads associated with mask negotiation, but the security of this approach relies on the assumption of non-collusion between the third-party trusted server and the aggregation server. The study [41] removed the trusted third party by combining secret sharing and HPRG and using secret sharing additive homomorphism to complete mask aggregation jointly by users. However, in this approach, users are required to secret share the seed and encrypt the secret share in each round and then send it to every other user; meanwhile, users are required to encrypt model parameters, and computing each global model parameter is required to solve the discrete logarithm problem, which leads to a performance bottleneck. The work [42] developed a single mask-based chained aggregation structure, where the head node user masks the model using an initial random mask sent by the aggregation server and then sends the masked model as a random mask to the next user, and finally, the server removes the initial mask to aggregate the global model. However, the efficiency of this structure relies on the performance and stability of each user, and when users drop

out of the network or when the latency is too high, the approach may fail to aggregate a valid global model.

In this paper, an efficient random mask privacy aggregation method is proposed based on EC-ElGamal, HRPG, and CRT, which can aggregate random masks efficiently and protect mask privacy without adding trusted third parties and communication between users, and the method handles dropped out users without introducing huge computational and communication overheads. In the case of no user dropout, the aggregation is completed with only two rounds of communication.

3. Preliminaries

3.1. EC-ElGamal Homomorphic Encryption

EC-ElGamal [43] is an implementation of the ElGamal encryption algorithm [44] on the elliptic curve (EC) with additive homomorphism. The security of EC-ElGamal relies on the elliptic curve discrete logarithm problem (ECDLP) on EC, and it is IND-CPA secure under the decisional Diffie–Hellman (DDH) assumption. Its encryption scheme with additive homomorphism is described below:

Initialization. Choose a large prime p and then choose a, b in $GF(p)$ to determine an elliptic curve $E_p(a,b)$. Choose a point $G(x,y)$ on $E_p(a,b)$ as the base point, and the order of the point $G(x,y)$ is n. n is required to be a large prime.

Keygen. Randomly choose an integer $d(0 < d < n)$ as the private key sk, and then the public key is:

$$pk = skG \tag{1}$$

Encryption. $(C_1, C_2) \leftarrow ECE.Enc(P_m, k_i, pk)$. Assuming P_m is the plaintext, randomly select $k \in Z_n$ and compute the ciphertext (C_1, C_2):

$$C_1 = k_iG, C_2 = P_mG + k_ipk \tag{2}$$

Decryption. $P_m \leftarrow ECE.Dec(C_2, C_1, sk)$. To decrypt (C_1, C_2), compute $m = C_2 - skC_1$:

$$P_mG = P_mG + k_ipk - skk_iG \tag{3}$$

By addressing the ECDLP, the plaintext P_m can be decrypted eventually.

Additive homomorphism. EC-ElGamal encryption has additive homomorphism. Suppose that the ciphertexts of plaintext P_{m_1}, P_{m_2} are $(k_iG, P_{m_1}G + k_ipk)$,$(k_iG, P_{m_2}G + k_ipk)$, respectively. The ciphertexts are added together:

$$(k_iG + k_iG, (P_{m_1} + P_{m_2})G + 2k_ipk) \tag{4}$$

When using sk decryption to obtain $(P_{m_1} + P_{m_2})G$ and the plaintext $P_{m_1} + P_{m_2}$, it is also required to address the ECDLP. The research papers [45,46] proposed to improve computational efficiency by converting an intractable ECDLP (i.e., a large integer) into several smaller ECDLP by utilizing the CRT and proved that this approach does not compromise security.

3.2. Homomorphic Pseudorandom Generator

A pseudorandom function (PRF) is a keyed function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$. For a random key $k \in \mathcal{K}$, the output of the function $F(k, \cdot)$ is indistinguishable from the output of a uniformly random function when given black-box access. Homomorphic pseudorandom function (HPRF) [47] is a pseudorandom number function with homomorphic properties. For any key $k, k \in \mathcal{K}$ and any input $x \in \mathcal{X}$, there is $f(k_1 + k_2, x) = f(k_1, x) \oplus F(k_2, x)$. A pseudorandom generator (PRG) is an efficiently computable function $G : \mathcal{S} \to \mathcal{R}$. The distribution of $G(s)$ is computationally indistinguishable from the distribution of r for a uniform s in $\mathcal{S}$ and a uniform r in $\mathcal{R}$. Similarly, for any $G(s_1, x)$ and $G(s_2, x)$ that satisfy $G(s_1 + s_2, x) = G(s_1, x) \oplus G(s_2, x)$, the PRG function $G : \mathcal{S} \to \mathcal{R}$ is called a seeded HPRG.

A simple secure-key homomorphic PRF can be constructed in the Random Oracle Model (ROM). Suppose that G is a finite cyclic group of prime order q. $H : \mathcal{X} \to G$ is a hash function modeled as a random oracle, and $F : Z_q \times \mathcal{X} \to G$ is constructed as:

$$F(k, x) = k \cdot H(x) \tag{5}$$

$F(k, x)$ satisfies additive homomorphism:

$$F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) \tag{6}$$

This construction was first proposed and proved in the work [48]. In the ROM, it is assumed that DDH holds in G and that F is a secure PRF. The paper [49] uses this HPRF in key rotation for authenticated encryption.

Based on the above HPRF, an HPRG can be constructed as $HPRG(s, x) = s \cdot H(x)$, which satisfies:

$$HPRG(s_1 + s_2, x) = HPRG(s_1, x) + HPRG(s_2, x) \tag{7}$$

3.3. Secret Sharing

(t, n)-secret sharing was proposed by Shamir and Blakey in 1979 [50]. Secret s is divided into n sub-secrets, each of which is held by a user; more than or equal to t users can reconstruct secret s, while less than t users cannot reconstruct the secret and do not obtain any information on s. For a secret s, (t, n)-secret sharing is performed in the following steps:

Initialization. Secret holder u_s randomly selects $t-1$ positive integers $a_1, \ldots, a_{t-1}$ from Z_P while making $a_0 = s$. Based on the above values, construct a polynomial with the highest order of $t-1$:$f(x)_s = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1} mod\ P$.

Secret Sharing. The secret holder randomly selects n points $x_1, x_2, \ldots, x_n$ on the polynomial $f(x)$, then computes the function values $f(x_1), f(x_2), \ldots, f(x_n)$ corresponding to each point, and finally share $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n))$ as a sub-secret to the other users. The formal representation is $(x_i, f(x_i)_s)_{i \in n} \leftarrow ASS.share(s, t, n)$.

Secret Reconstruction. Given any no less than t sub-secrets $(x_i, f(x_i)_s)$, the polynomial $f(x)_s$ can be determined by Lagrange interpolation. When $x = 0$, the constant term a_0 can be computed as the secret s, and the formal representation is $s \leftarrow ASS.recon\big((x_i, f(x_i)_s)_{i \in n}, t\big)$.

Homomorphism. Shamir secret sharing has additive homomorphism. Given s_1, s_2 and secret sharing (t, n) and distributing the secret share to other users, the holders of the sub-secrets can locally compute $\left(x_i, f(x_i)_{s_1} + f(x_i)_{s_2}\right)_{i \in n}$. Since the highest order of the result of adding two $t-1$th polynomials is still $t-1$, given any (no less than t) sub-secretions $\left(f(x_i)_{s_1} + f(x_i)_{s_2}\right)_{i \in n}$, we have $f(x) = f(x)_{s_1} + f(x)_{s_2}$; and when $x = 0$, we have $s_1 + s_2 = f(0)_{s_1} + f(0)_{s_2}$.

4. Proposed Scheme

4.1. Threat Model and Goals

In this paper, participants are divided into two types: users and aggregation servers. The basic structure and the aggregation process are illustrated in Figure 1. The aggregation server S is responsible for aggregating local models. Each user $u_i \in \mathcal{U}$ has a local private dataset, representing a client, which obtains the local model $\boldsymbol{m_i}$ by training on the local dataset and uploading it to the aggregation server. Other notations used in this paper are listed in Table 1.

Threat model: This paper considers semi-honest threat models commonly used in FL, where aggregation servers and users may be semi-honest, i.e., they follow the rules for training but are curious about the local models of honest users, intending to infer the honest user's private information by observing their local models. It should be noted that malicious servers forging aggregation results and malicious users implementing poisoning attacks are not considered in this paper. This requires additional techniques to implement

security when participants are malicious, such as integrity verification techniques and how to compute model similarity when models are protected.

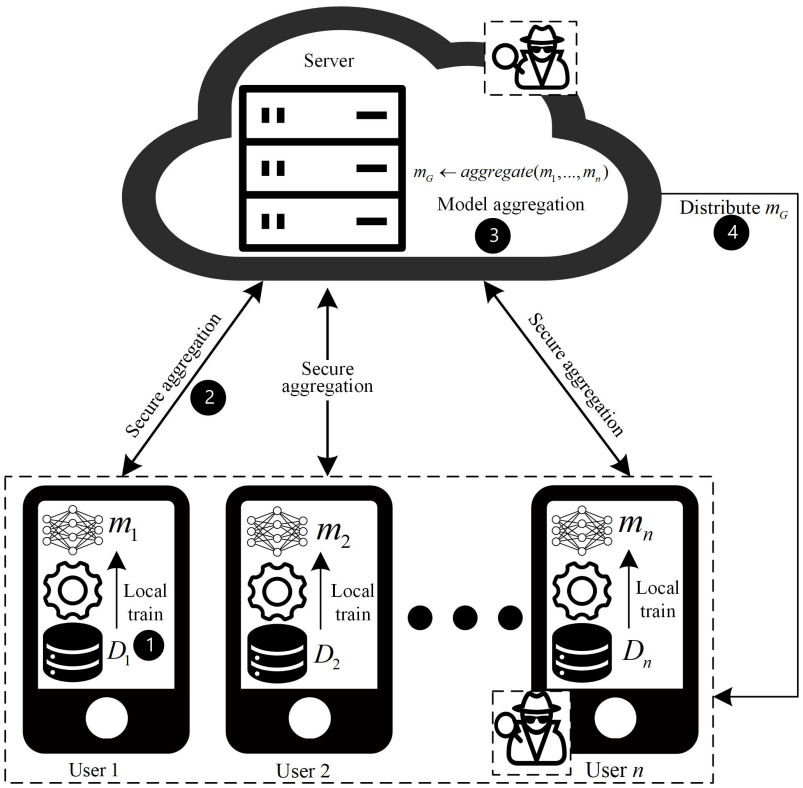


Figure 1. FL model aggregation.

Table 1. Symbols and description.

Symbols	Description
u_i	user i
S	aggregation server S
$\boldsymbol{m_i}$	local model of user u_i
$\boldsymbol{m_G}$	global model
L	user input vector size
s_i	random seed of user u_i
$\boldsymbol{r_i}/\boldsymbol{R}$	random mask of user u_i/sum of random mask
d	number of dropout users
$\mathcal{U}, \mathcal{U}_1, \cdots$	collection of users in different stages
n	number of users
t	threshold
$a_1^i, a_2^i, \cdots$	congruent coefficient of user u_i
pk/sk	EC-ElGamal public–private key

In addition to the semi-honest setting, this paper also considers the collusion between the server and the users, where the server and the colluding users attempt to increase the success rate of stealing the honest user's private information by sharing the internal state, and the received and the sent messages. For collusion scenarios, the scheme proposed in this paper aims to guarantee the privacy of the honest user's model, i.e., in cases where less than t users and servers collude, they do not learn any information about the honest

user's local model. Note that this paper does not consider the case $t = |\mathcal{U}| - 1$, where the collusion users and the server can compute the honest user's model information from the aggregated results by making a difference. Furthermore, for the setting of the value of t, in cross-silo FL, i.e., without considering the user dropout, the maximum number of collusions our scheme can tolerate is $t = |\mathcal{U}| - 2$. In cross-device FL, users may drop out from the training at any time, and our scheme supports this; in this case, the maximum number of colluding users our scheme can tolerate depends on the threshold for secret sharing t. Reducing the tolerance in a cross-device FL setting is justified because the probability of large-scale user collusion is greatly reduced with a large number of users and a wide geographic distribution.

Design goals: Based on the above threat model and considering the requirements of the FL application scenario, the design goals of this paper are:

Privacy: Ensure that semi-honest servers and users cannot obtain any private information about the local models of honest users.

Efficiency: The protection of the model does not incur excessive computational and communication overheads to the user, and the overall aggregation scheme is efficient.

Dropout Tolerance: The ability to tolerate user dropouts during the training process.

4.2. Scheme Overview

4.2.1. Model Mask

Many studies have proposed using techniques such as DP and HE to protect local models, but these techniques either incur huge computational and communication overheads on users or require considering the trade-off between privacy level and model quality. This paper achieves model protection based on a simple and efficient One-time Pad [51] (OTP). A random mask is taken as a one-time key to mask the model, and due to the randomness of the mask, the distribution of the model parameters will be disrupted. Based on this, privacy attacks against the local model can be effectively prevented, and the adversary cannot recover the model parameters when the random mask is not leaked. Meanwhile, only one additive operation is involved in protecting the model, which guarantees both high efficiency and no compromise of security. In SecAgg, a pairwise mask protection model is used to ensure that the masks of all users can cancel each other after aggregation, but this requires much communication to negotiate the pairwise masks, and the aggregation time increases substantially with the number of dropped out users.

To avoid problems such as computational and communication overheads associated with pairwise masks, this paper uses independent masks to protect the model. User u_i randomly selects a seed s_i, then uses a random mask generated by PRG for $\boldsymbol{r_i}$, and finally, masks the local model with $\boldsymbol{r_i}$:

$$\widetilde{\boldsymbol{m}}_{\boldsymbol{i}} = \boldsymbol{m_i} + \boldsymbol{r_i} \tag{8}$$

Then, user u_i sends $\widetilde{m}_i$ to the server for aggregation.

4.2.2. Model Aggregation

Assuming that the server knows the sum $\boldsymbol{R} = \sum_{u_i \in \mathcal{U}} \boldsymbol{r_i}$ of the user's random mask, after receiving the user's model $\widetilde{\boldsymbol{m}}_{\boldsymbol{i}}, i \in \mathcal{U}$, the server can aggregate the global model:

$$\boldsymbol{m_G} = \sum_{u_i \in \mathcal{U}} \widetilde{\boldsymbol{m}}_{\boldsymbol{i}} - \boldsymbol{R} = \sum_{u_i \in \mathcal{U}} \widetilde{\boldsymbol{m}}_{\boldsymbol{i}} - \sum_{u_i \in \mathcal{U}} \boldsymbol{r_i} = \sum_{u_i \in \mathcal{U}} \boldsymbol{m_i} \tag{9}$$

Since the random mask is completely canceled, the global model is almost unaffected by the mask, and the server cannot obtain the local model of a single user during the aggregation process, which guarantees the local model privacy.

In the above process, this paper assumes that the server knows $\boldsymbol{R}$. The most direct way is that user u_i sends the mask $\boldsymbol{r_i}$ to the server, but the server is semi-honest and can compute the user's local model when it knows $\widetilde{\boldsymbol{m}}_{\boldsymbol{i}}$ and $\boldsymbol{r_i}$, which leads to privacy risks.

Therefore, it is necessary to guarantee that the server computes $\boldsymbol{R}$ without knowing the individual user $\boldsymbol{r_i}$. The research works [39,40] introduced a third party to aggregate the masks, which requires that the third party is trusted and does not collude with the aggregation server. The study [41] combined HRPG and secret sharing to aggregate the masks, which requires the user to communicate with all the other users once to share the secret share. Aiming at resolving the above issues, this paper proposes an efficient random mask secure aggregation (ERMSA) method that can tolerate user dropouts.

4.3. ERMSA

4.3.1. Multi-Key Privacy Aggregation

To correctly compute $\boldsymbol{R}$ without exposing $\boldsymbol{r_i}$ to the server, this paper proposes a multi-key privacy aggregation algorithm based on EC-ElGamal's homomorphism and ciphertext correlation. The algorithm allows the user to encrypt the input using a public key and a temporary random key k_i and then implements the key transformation through one interaction with the server. Finally, the server can decrypt the sum of the user's input using its own private key. In this algorithm, the k_i is generated temporarily and randomly by each user and does not need to be shared among users, thereby avoiding possible security risks associated with key sharing.

Specifically, before the aggregation process begins, users $u_i, u_i \in \mathcal{U}$ locally generate a pair of public–private keys sk_i, pk_i, the server generates the public–private keys sk_s, pk_s, and then each user sends the public key to the server, which computes the global public key PK:

$$PK = \sum_{u_i \in \mathcal{U}} pk_i \tag{10}$$

The server sends PK and pk_s to each user.

As shown in Algorithm 1, after obtaining the global public key PK, the user u_i randomly generates a large integer k_i as a temporary key and then encrypts the input v_i using PK and k_i according to Equation (2):

$$C_{1,u_i} = k_i G, C_{2,u_i} = v_i G + k_i PK \tag{11}$$

Then, the user sends the ciphertext $\left(C_{1,u_i}, C_{2,u_i}\right)$ to the server, which computes the ciphertext based on the additive homomorphism of EC-ElGamal:

$$\begin{aligned} C_{1,s} &= \sum_{u_i \in \mathcal{U}} k_i G \\ C_{2,s} &= \sum_{u_i \in \mathcal{U}} v_i G + \sum_{u_i \in \mathcal{U}} k_i PK \end{aligned} \tag{12}$$

The server sends $C_{1,s}, C_{2,s}$ to each user, and user u_i computes it locally:

$$C'_{2,u_i} = -C_{1,s} sk_i + k_i pk_s \tag{13}$$

Then, each user sends (C'_{1,u_i}, C'_{2,u_i}) to the server, where $C'_{1,u_i} = C_{1,u_i}$. The server receives all users' (C'_{1,u_i}, C'_{2,u_i}) and calculates:

$$C'_{2,s} = C_{2,s} + \sum_{u_i \in \mathcal{U}} C'_{2,u_i} \tag{14}$$

The result is $C'_{2,s} = \sum_{u_i \in \mathcal{U}} v_i G + K pk_s$, where $K = \sum_{u_i \in \mathcal{U}} k_i$, and the detailed derivation is shown in Appendix A. The server uses sk_s to decrypt $\left(C'_{1,s}, C'_{2,s}\right)$ to obtain $\sum_{u_i \in \mathcal{U}} v_i G$, where $C'_{1,s} = C_{1,s} = KG$. To obtain $V = \sum_{u_i \in \mathcal{U}} v_i$, the ECDLP needs to be solved, and it is possible to compute V when the values of the elements in v_i are small and the dimension of v_i is low; however, when the values of the elements in v_i are large and the dimension of v_i

is high, it will greatly reduce the efficiency of Algorithm 1, and this issue will be addressed in Section 4.3.2.

Algorithm 1 ensures that the server can compute the sum V of the user inputs without exposing the user input v_i. Even in the case of $N-2$ users colluding with the server, the algorithm still ensures the privacy of the honest user's inputs.

Algorithm 1 Multi-Key Privacy Aggregation Algorithm.

Input:
Each user u_i public–private key, sk_i, pk_i, v_i, PK, Server S public–private key, sk_s, pk_s;
Output:
The sum of all user inputs V;

1: Phase I
2: User u_i generates random temporary key k_i, encrypts input v_i with PK: $\left(C_{1,u_i}, C_{2,u_i}\right) \leftarrow ECE.Enc(v_i, k_i, PK)$.
3: User u_i sends result $\left(C_{1,u_i}, C_{2,u_i}\right)$ to the server.
4: Server S receives $\left(C_{1,u_i}, C_{2,u_i}\right)$ sent by user u_i and calculates: $\left(C_{1,s}, C_{2,s}\right) \leftarrow \left(\sum_{u_i \in \mathcal{U}_1} C_{1,u_i}, \sum_{u_i \in \mathcal{U}_1} C_{2,u_i}\right)$.
5: Server S send the result of the calculation $\left(C_{1,s}, C_{2,s}\right)$ to each user;
6: Phase II
7: User u_i receives the server calculation $\left(C_{1,s}, C_{2,s}\right)$ and then calculates: $C'_{2,u_i} = -C_{1,s} sk_i + k_i pk_s$.
8: User u_i send the results of the calculation to the server.
9: Server receives $\left(C'_{1,u_i}, C'_{2,u_i}\right)$ from user u_i and calculates: $C'_{2,s} = C_{2,s} + \sum_{u_i \in \mathcal{U}} C'_{2,u_i}$.
10: Server decrypts $\left(C'_{1,s}, C'_{2,s}\right)$ using private key sk_s to obtain: $V = ECE.Dec\left(C'_{1,u_i}, C'_{2,u_i}, sk_s\right)$;

4.3.2. Random Mask Efficient Aggregation

By taking the mask $\boldsymbol{r_i}$ as input to Algorithm 1, the user can allow the server to compute $\boldsymbol{R}$ and correctly aggregate the global model without exposing $\boldsymbol{r_i}$. However, this will lead to two problems: (1) Since the dimension of $\boldsymbol{r_i}$ is the same as that of the model, the user encrypts the mask $\boldsymbol{r_i}$ directly, which requires large computation and communication overheads when dealing with high-dimensional deep FL models, and under cross-silo FL, where most of the clients are mobile devices, the hardware may not meet the training requirements. (2) When the server decrypts the ciphertext to obtain $\boldsymbol{R}$, the ECDLP needs to be solved. When the dimension of v_i is too high and the values of the elements are too large, the decryption time increases significantly.

To address the above issues and achieve efficient FL secure aggregation, this paper introduces HRPG and CTR to achieve efficient random mask privacy aggregation. Specifically, for Problem 1, considering that $\boldsymbol{r_i}$ is a random number, this paper modifies the random mask generation method to avoid the huge computation and communication overheads incurred by users encrypting each element of the high-dimensional mask. The user employs HRPG introduced in Section 3.2 to generate the random mask, and through HRPG, the aggregation of the mask $\boldsymbol{r_i}$ is converted to the aggregation of the seed s_i, where $1 = |s_i| \ll |\boldsymbol{r_i}|$. Thus, the user only needs to encrypt the seed s_i once without introducing too much computation and communication overheads. Meanwhile, the number of times the server needs to solve the ECDLP is reduced to one. For Problem 2, this paper improves the aggregation efficiency by converting a difficult ECDLP into several simple ECDLPs via CRT for solving.

As shown in Equation (3), when all users share an x, additive homomorphism is satisfied for any $s \in \mathcal{K}$. Considering that $HRPG(s, x)$ is fixed when x and s are fixed; in this case, all model parameters are masked using the same random mask, and the adversary can reconstruct the original model parameters easily through enumeration attacks with the range of model parameters determined. To address this issue, this paper transforms x to generate different masks for different model parameters with s determined; to avoid the

extra communication overhead caused by negotiating x between users, this paper adopts training rounds and model parameter index splicing to construct a shared sequence x and then uses HPRG to generate the mask $\boldsymbol{r_i}$ based on the x and s.

Firstly, user $u_i \in \mathcal{U}$ locally selects a seed $s_i \in \mathcal{K}$ at random and calculates:

$$\boldsymbol{r_i} = HPRG(s_i) \tag{15}$$

Specifically, $HPRG(s_i) = [F(s_i, b||\tau)], 0 < b < L$, b is the index of each model parameter, L denotes the number of model parameters, τ denotes the number of current training rounds, and F is:

$$F(s_i, b||\tau) = s_i \cdot H(b||\tau) \tag{16}$$

H is the hash function. According to the properties of HPRG, we have:

$$\boldsymbol{R} = HPRG\left(\sum_{u_i \in \mathcal{U}} s_i\right) = \sum_{u_i \in \mathcal{U}} HPRG(s_i) = \sum_{u_i \in \mathcal{U}} \boldsymbol{r_i} \tag{17}$$

By using Algorithm 1 to aggregate seed s_i, the server, after calculating $\sum_{u_i \in \mathcal{U}} s_i$ using HRPG, can calculate $\boldsymbol{R}$. In the whole process, the user needs to perform encryption only once and a single ciphertext transmission, and the server needs to perform decryption only once.

To further reduce the server decryption time and improve the overall aggregation efficiency, this paper converts a difficult ECDLP into several simple ECDLP via CRT encoding s_i for solving.

Firstly, the user represents the seed s_i by its several congruencies a_i:

$$\begin{aligned} s_i &\equiv a_1 \ mod \ p_1 \\ s_i &\equiv a_2 \ mod \ p_2 \\ &\cdots \\ s_i &\equiv a_n \ mod \ p_n \end{aligned} \tag{18}$$

where $a_i \ll s_i, p_2, \ldots, p_n$ are mutually prime and shared by all participants. According to the CRT, when $a_1, a_2, \ldots, a_n$ are known, the unique solution to the above equation, s_i, can be obtained as:

$$s_i \equiv \sum_{i=1}^{n} a_i \times \frac{N}{p_i} \left[\left(\frac{N}{p_i} \right)^{-1} \right]_{p_i} (mod \ N), N = \prod_{i=1}^{n} p_n \tag{19}$$

The CRT has additive homomorphism, with a list of congruent cosets $a_1^i, a_2^i, \ldots, a_n^i$ and $a_1^j, a_2^j, \ldots, a_n^j$ when the two seeds s_i and s_j are known:

$$\begin{aligned} s_i + s_j &= \sum_{k=1}^{n} a_k^i \times \frac{N}{p_i} \left[\left(\frac{N}{p_i} \right)^{-1} \right]_{p_i} (mod \ N) \\ &+ \sum_{k=1}^{n} a_k^j \times \frac{N}{p_i} \left[\left(\frac{N}{p_i} \right)^{-1} \right]_{p_i} (mod \ N) \\ &= \left(\sum_{k=1}^{n} a_k^i + a_k^j \right) \times \frac{N}{p_i} \left[\left(\frac{N}{p_i} \right)^{-1} \right]_{p_i} (mod \ N) \end{aligned} \tag{20}$$

Based on the above properties, each user u_i CRT encodes the seed s_i:

$$CRT[s_i] = a_1^i, \ldots, a_n^i \tag{21}$$

Each user u_i takes the congruent residues $a_1^i, \ldots, a_n^i$ as inputs to Algorithm 1, then the server computes $\sum_{u_i \in \mathcal{U}} a_1^i, \ldots, \sum_{u_i \in \mathcal{U}} a_n^i$, respectively, and finally, based on the homomorphisms of the CRT, the server decodes to obtain $\sum_{u_i \in \mathcal{U}} s_i$.

In the above process, the user encrypts each congruent cosine number, which incurs some computation and communication overhead compared to encrypting only s_i. When using NIST P-256 elliptic curves, a single EC-ElGamal encryption takes about 1.5–2 ms and the ciphertext size is about 0.125 kb. Assuming that the number of congruencies is 10, the computation time consumed by the user encryption is approximately 15–20 ms, and the communication overhead is approximately 1.25 kb. It can be seen that the user's added overhead is not significant, but this improves server decryption efficiency and reduces overall aggregation time greatly.

Additionally, to further reduce the computation overhead during training, the masks $\boldsymbol{r_i}$, $\frac{N}{p_i}$, and $\left[\left(\frac{N}{p_i}\right)^{-1}\right]_{p_i}$ can be computed offline.

4.3.3. Tolerate User Dropouts

Section 4.3.2 has introduced the implementation of efficient random mask privacy aggregation that enables efficient FL security aggregation, but it cannot tolerate user dropouts. Since random mask privacy aggregation relies on Algorithm 1, if a user drops out, the server cannot receive the $\left(C'_{1,u_i}, C'_{2,u_i}\right)$ computed by that user, and it will fail to complete the key conversion and decryption correctly.

Thus, the above random mask efficient privacy aggregation scheme can only be used in cross-silo FL. In cross-device FL, clients may drop out at any time during the training process due to network state, energy constraints, and other issues. To solve this problem, this paper introduces secret sharing to enable the approach to tolerate user dropouts at any time, and the FL training can be performed normally when a certain number of users drop out.

Specifically, before the training process begins, user u_i shares sk_i secretly after generating sk_i: $\left(j, sk_{i,j}\right)_{u_j \in \mathcal{U}} \leftarrow SS.share(sk_i, t, |\mathcal{U}|)$, where t denotes the threshold and $\left(j, sk_{i,j}\right)$ denotes the secret share shared by u_i to u_j. To share the secret share securely, user u_i obtains the public key pk_j of other users via the server, then performs key agreement, and finally, encrypts the secret share using the agreement key and the AES algorithm and sends the ciphertext to the aggregation server. After receiving the secret share, the aggregation server forwards it to other users. The user decrypts the secret share ciphertext shared by other users using the key agreement and keeps the secret share locally.

In this paper, it is assumed that the user may be dropped at any time during the training process. As shown in Algorithm 1, the surviving user $u_i, u_i \in \mathcal{U}_1$ in the first phase successfully sent C_{1,u_i}, C_{2,u_i} to the server. The user $u_i, u_i \in \mathcal{U}_2$ who survived the second phase successfully sent C'_{1,u_i}, C'_{2,u_i} to the server, where $\mathcal{U}_2 \subseteq \mathcal{U}_1 \subseteq \mathcal{U}$. At this time, the server calculates $(C_{1,s}, C_{2,s})$ and $\left(C'_{1,s}, C'_{2,s}\right)$, respectively:

$$\begin{gathered}\left(\sum_{u_i \in \mathcal{U}_1} k_i G, \sum_{u_i \in \mathcal{U}_1} v_i G + \sum_{u_i \in \mathcal{U}_1} k_i PK\right) \\ \left(\sum_{u_i \in \mathcal{U}_2} C'_{1,u_i}, C_{2,s} + \sum_{u_i \in \mathcal{U}_2} C'_{2,u_i}\right)\end{gathered} \tag{22}$$

Since user $u_j, u_j \in \mathcal{U}_1 \setminus \mathcal{U}_2$ drops out, the server does not receive C'_{2,u_j} computed by the dropped out user, and thus, it cannot perform the key conversion and then decryption

through calculation, following Equation (14) (the detailed derivation process is provided in Appendix B). At this point, the server calculates $C'_{1,s}, C'_{2,s}$ using the following equation:

$$\begin{aligned} C'_{1,s} &= \sum_{u_i \in \mathcal{U}_2} k_i G \\ C'_{2,s} &= C_{2,s} + \sum_{u_i \in \mathcal{U}_2} C'_{2,u_i} + C_{1,s} \sum_{u_i \in \mathcal{U} \setminus \mathcal{U}_2} -sk_i \end{aligned} \tag{23}$$

The server needs to calculate $\sum_{u_i \in \mathcal{U} \setminus \mathcal{U}_2} -sk_i$. Users share the secret sk_i with other users before training, and if user $u_j, u_j \in \mathcal{U} \setminus \mathcal{U}_2$ drops out during training, the other user u_i will sum up all the secret shares shared by all the users who drop out and send them to the server: $\left(i, \sum_{j \in \mathcal{U} \setminus \mathcal{U}_2} sk_{j,i}\right)_{u_i \in \mathcal{U}_2}$. Then, the server can recover $\sum_{u_i \in \mathcal{U} \setminus \mathcal{U}_2} sk_j$: $\sum_{u_i \in \mathcal{U} \setminus \mathcal{U}_2} sk_j \leftarrow SS.recon\left(\left(i, \sum_{j \in \mathcal{U} \setminus \mathcal{U}_2} sk_{j,i}\right)_{u_i \in \mathcal{U}_2}, t\right)$. The above process leverages the secret sharing additive homomorphism, and the server only needs to perform one reconstruction to recover the sum of the dropped out users' private keys.

Additionally, when only one user drops out, the server ends up reconstructing this user's private key, but this does not compromise the privacy of that user's local model. The user's seed is encrypted using PK instead of his private key unless the server knows all the other users' sk_i and then computes the private key corresponding to PK. Since our scheme achieves tolerance for user dropouts through secret sharing, the number of tolerable users that can conspire with the server is reduced to a threshold $t-1$. This is because when more than t users collude, they can recover the private keys of all users, and by colluding with the server, they can compute the local model of the honest user.

4.4. ERMSA-Based Secure Aggregation

Our aggregation approach runs between a server S and n users, where the set of users is denoted as $\mathcal{U}$, and different phases of the set are distinguished by numerical subscripts, e.g., $\mathcal{U}_1$. For the convenience of representation, this paper sets unique identifiers, e.g., $1, 2, \ldots, n$, for each client, and distinguishes the user parameter by subscripts. Each user has a local model $\boldsymbol{m_i}$ of length L with each element ranging in Z_q. Furthermore, the subscript s is used to denote the server parameters, and the server is responsible for message forwarding between clients and the necessary computational services.The aggregation server communicates directly with the users to transfer data. The user–server interaction process is illustrated in Figure 2, where solid lines represent required interactions and dashed lines represent the presence of interactions that require additional interactions when the user drops out.

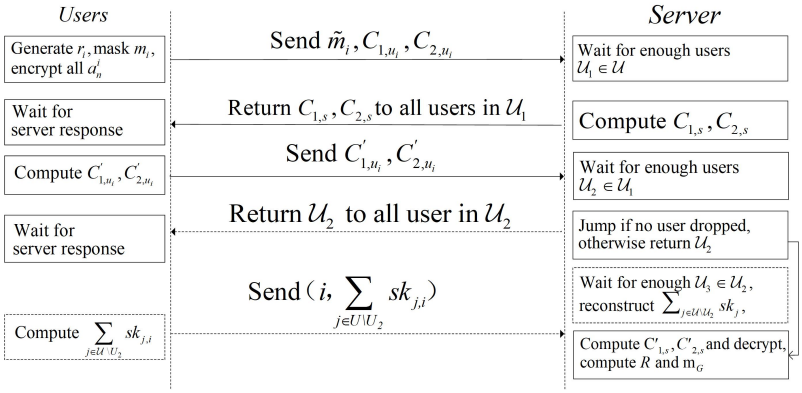


Figure 2. Interaction process.

The interaction between the user and the server is synchronized; in each round, the server waits for the user to send a message for a limited time, and the user is considered to have dropped out if he does not send a message beyond the limited time. Users can

drop out at any time, and the server can aggregate correctly if it receives messages sent by more than t users in each round and aborts aggregation immediately if fewer than t users survive in a certain round; moreover, it is assumed that the server stays online all the time and that the users and the server communicate using a secure channel.

The details of the proposed scheme are presented in Table 2, which shows the negotiation between the server and the user during the setup phase before the training starts, involving PK, threshold t, and mutual prime numbers $p_1, \ldots, p_n$, etc. The underlined parts in Table 2 indicate the operations that need to be performed by the user and the server in cross-device FL. Even in cross-device FL, the Unmask Model phase is not required to be executed without user dropout, and the server can complete the aggregation in the key conversion phase.

Table 2. ERMSA-based secure aggregation in FL.

ERMSA-based secure aggregation in FL

Setup:

- Each user u_i generates the public–private key sk_i,pk_i, and the server S generates the public–private key sk_s,pk_s; each user sends the pk_i to the aggregation server S; the server computes the public public key $PK = \sum_{u_i \in \mathcal{U}} pk_i$, and then it sends the PK, the pk_s, the threshold $t, p_1, \ldots, p_n, n \geq 2$, and the pk_i to each user u_i.
- Each user u_i secret-shares sk_i, $\left(j, sk_{i,j}\right)_{u_j \in \mathcal{U}} \leftarrow$ **SS.share**$(sk_i, t, |\mathcal{U}|)$, then generates a negotiation key $ssk_{(i,j)}$ using pk_j of sk_i and every other user u_j, encrypts the secret share by the AES algorithm, and finally sends the ciphertext to the server. The server forwards the secret share ciphertext to other users.
- All users u_i decrypt using $ssk_{(i,j)}$ to obtain $(i, sk_{1,i}), (i, sk_{2,i}), \ldots, (i, sk_{n,i})$.

Round 0 (**Collecting masked model**):

User u_i:

- Randomly select a seed s_i and a temporary key k_i to generate a random mask via **HPRG**: $\boldsymbol{r_i} =$ **HPRG**(s_i).
- Mask $\boldsymbol{m_i}$ using $\boldsymbol{r_i}$: $\widetilde{\boldsymbol{m}}_{\boldsymbol{i}} = \boldsymbol{m_i} + \boldsymbol{r_i}$
- CRT-encode the seed s_i: **CRT**$[s_i] = a_1^i, \ldots, a_n^i$
- Encrypt a_n^i using k_i and PK: $\left(C_{1,u_i}, C_{2,u_i}\right) \leftarrow$ **ECE.Enc**(a_n^i, k_i, PK)(encrypt all congruent ciphertexts identically using $\left(C_{1,u_i}, C_{2,u_i}\right)$ to represent all congruent ciphertexts)
- Send $\widetilde{\boldsymbol{m}}_{\boldsymbol{i}}$ and $\left(C_{1,u_i}, C_{2,u_i}\right)$ to the server, and then wait.

Server:

- Receive $\left(C_{1,u_i}, C_{2,u_i}\right)$ from at least t users (denote the set of users by $\mathcal{U}_1, \mathcal{U}_1 \subseteq \mathcal{U}$); otherwise abort.
- Sum the received seed ciphertexts $\left(C_{1,u_i}, C_{2,u_i}\right), u_i \in \mathcal{U}_1$: $(C_{1,s}, C_{2,s}) \leftarrow \left(\sum_{u_i \in \mathcal{U}_1} C_{1,u_i}, \sum_{u_i \in \mathcal{U}_1} C_{2,u_i}\right)$.
- Send $(C_{1,s}, C_{2,s})$ to each user in $\mathcal{U}_1$, and then wait.

Round 1 (**Key Conversion**):

User u_i:

- Receive $(C_{1,s}, C_{2,s})$ and then compute $C'_{2,u_i} = C_{1,s} sk_i + k_i pk_s$.
- Send $\left(C'_{1,u_i}, C'_{2,u_i}\right)$ to the server, $(C'_{1,u_i} = C_{1,u_i})$, and then wait.

Server:

- Receive $\left(C'_{1,u_i}, C'_{2,u_i}\right)$ from at least t users (denote the set of users by $\mathcal{U}_2$ and $\mathcal{U}_2 \subseteq \mathcal{U}_1$); otherwise abort.
- If $\mathcal{U}_2 = \mathcal{U}$, no dropped users:
  - Compute $C'_{2,s}$: $C'_{2,s} = C_{2,s} + \sum_{u_i \in \mathcal{U}} C'_{2,u_i}$, and then use sk_s to decrypt: $\sum_{u_i \in \mathcal{U}_2} a_n^i \leftarrow$ **ECE.Dec**$\left(C'_{1,s}, C'_{2,s}, sk_s\right)$, $C'_{1,s} = C_{1,s}$.
  - Recover the sum of the seeds $\sum_{u_i \in \mathcal{U}} s_i =$ **CRT**$\left[\sum_{u_i \in \mathcal{U}} a_1^i, \ldots, \sum_{u_i \in \mathcal{U}} a_n^i\right]$ and calculate the mask $\boldsymbol{R}$: $\boldsymbol{R} = HPRG\left(\sum_{u_i \in \mathcal{U}_2} s_i\right)$.
  - Aggregate local models and remove masks: $\boldsymbol{m_G} = \sum_{u_i \in \mathcal{U}_2} \widetilde{\boldsymbol{m}}_i - \boldsymbol{R}$.
- Otherwise, send list $\mathcal{U}_2$ to each user in $\mathcal{U}_2$, and then wait.

Round 2 (**Unmask Model**):

User u_i:

- Receive the list of online users $\mathcal{U}_2$, and if $\mathcal{U}_2 \subseteq \mathcal{U}_1, |\mathcal{U}_2| \geq t$ send $\left(i, \sum_{u_j \in \mathcal{U} \setminus \mathcal{U}_2} sk_{j,i}\right)$ to server.

Server:

- Receive $\left(i, \sum_{u_j \in \mathcal{U} \setminus \mathcal{U}_2} sk_{j,i}\right)$ from at least t users (denote the set of users by $\mathcal{U}_3$ and $\mathcal{U}_3 \in \mathcal{U}_2$); otherwise abort.
- Recover the sum of the dropped user's private key $\sum_{u_j \in \mathcal{U} \setminus \mathcal{U}_2} sk_j$: $\sum_{u_j \in \mathcal{U} \setminus \mathcal{U}_2} sk_j \leftarrow$ **SS.recon**$\left(\left(i, \sum_{u_j \in \mathcal{U} \setminus \mathcal{U}_2} sk_{j,i}\right)_{u_i \in \mathcal{U}_3}, t\right)$.
- Compute $C'_{1,s}, C'_{2,s}$: $C'_{1,s} = \sum_{u_i \in \mathcal{U}_2} k_i G, C'_{2,s} = C_{2,s} + \sum_{u_i \in \mathcal{U}_2} C'_{2,u_i} + C_{1,s} \sum_{u_i \in \mathcal{U} \setminus \mathcal{U}_2} (-sk_i)$, and then use sk_i to decrypt: $\sum_{u_i \in \mathcal{U}_1} a_n^i \leftarrow$ **ECE.Dec**$\left(C'_{1,s}, C'_{2,s}, sk_s\right)$.
- Recover the sum of the seeds: $\sum_{u_i \in \mathcal{U}_1} s_i =$ **CRT**$\left[\sum_{u_i \in \mathcal{U}_1} a_1^i, \ldots, \sum_{u_i \in \mathcal{U}_1} a_n^i\right]$ and calculate the mask $\boldsymbol{R}$: $\boldsymbol{R} =$ **HPRG**$\left(\sum_{u_i \in \mathcal{U}_2} s_i\right)$.
- Aggregate local models and remove masks: $\boldsymbol{m_G} = \sum_{u_i \in \mathcal{U}_1} \widetilde{\boldsymbol{m}}_i - \boldsymbol{R}$.

5. Correctness and Security Analysis

5.1. Correctness

Theorem 1. *(Correctness) If the user follows the aggregation rule and* $|\mathcal{U}_3| \geq t, \mathcal{U}_3 \subseteq \mathcal{U}_2 \subseteq \mathcal{U}_1 \subseteq \mathcal{U}, \mathcal{U}' = \mathcal{U} \backslash \mathcal{U}_2$ *denotes the dropped out user, the server S can correctly calculate the sum of the user's inputs* $\sum_{u_i \in \mathcal{U}_1} \boldsymbol{m_i}$.

Proof of Theorem 1. When $|\mathcal{U}_3| \geq t$, the server receives no less than t sub-secret shares, and based on the homomorphism of secret sharing, the server can reconstruct $\sum_{u_i \in \mathcal{U}'} sk_i$. The server, knowing that $\sum_{u_i \in \mathcal{U}'} sk_i$, according to Equation (23), can calculate $C'_{1,s} = \sum_{u_i \in \mathcal{U}_2} k_i G$, $C'_{2,s} = \sum_{u_i \in \mathcal{U}_1} a_n^i G + \sum_{u_i \in \mathcal{U}_2} k_i pk_s$, decrypt by sk_s to obtain $\sum_{u_i \in \mathcal{U}_1} a_1^i, \ldots, \sum_{u_i \in \mathcal{U}_1} a_n^i$, and calculate from the homomorphism of CRT to obtain $\sum_{u_i \in \mathcal{U}_1} s_i$.

Since users use HPRG to generate random masks, we have:

$$HPRG\left(\sum_{u_i \in \mathcal{U}_1} s_i\right) = \sum_{u_i \in \mathcal{U}_1} HPRG(s_i) \tag{24}$$

The server can calculate:

$$\begin{aligned} \boldsymbol{m_G} &= \sum_{u_i \in \mathcal{U}_1} \widetilde{\boldsymbol{m}}_{\boldsymbol{i}} - HPRG\left(\sum_{u_i \in \mathcal{U}_1} s_i\right) \\ &= \sum_{u_i \in \mathcal{U}_1} \left(\widetilde{\boldsymbol{m}}_{\boldsymbol{i}} - HPRG(s_i)\right) \\ &= \sum_{u_i \in \mathcal{U}_1} \left(\boldsymbol{m_i} + HPRG(s_i) - HPRG(s_i)\right) \\ &= \sum_{u_i \in \mathcal{U}_1} \boldsymbol{m_i} \end{aligned} \tag{25}$$

□

5.2. Security

This section will show that our proposed scheme is secure in a semi-honest and curious setting. When aggregating using a threshold t, regardless of when the user dropout occurs, the server who colludes with any less than t users cannot infer anything about the inputs of the honest users except that they know the outputs, the seeds' sums, and their own inputs. The scheme involves an aggregation server S and n users. Specifically, the set of users is denoted by $\mathcal{U}$, the underlying cryptographic primitive is instantiated with the security parameter κ, and the users have a local model $\boldsymbol{m_i}$, a seed s_i, and a mask $\boldsymbol{r_i}$. Furthermore, the set of $\boldsymbol{m_i}$ users in any subset of users $\mathcal{U}$ is denoted as $m_{\mathcal{U}}$.

Let the set of semi-honest users and semi-honest servers be $\mathcal{C} \subseteq \mathcal{U} \cup S$. In the case of $|\mathcal{C} \backslash S| < t$, for any set $\mathcal{C}$, $REAL_{\mathcal{C}^{\mathcal{U},t,\kappa}}(m_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3)$ is a random variable representing the user in $\mathcal{C}$ in the combined view when the above scheme is executed, and its randomness exceeds the internal randomness of all participants and the randomness of the initialization phase.

For the collusion issue, this paper considers two cases: (1) only semi-honest users collude and the server does not collude with the users, and (2) semi-honest users and servers collude. In the following, security statements and proofs are provided for each of the two cases.

Theorem 2. *(Security against the collusion of only half-honest users) There exists a probabilistic polynomial time (PPT) simulator SIM for all* $\mathcal{U}, t, k, m_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3$, *where* $\mathcal{U} \supseteq \mathcal{C}, |\mathcal{C}| < t, \mathcal{U} \supseteq \mathcal{U}_1 \supseteq \mathcal{U}_2 \supseteq \mathcal{U}_3$. *The output of SIM is computationally indistinguishable from that of* $REAL_{\mathcal{C}^{\mathcal{U},t,\kappa}}$:

$$\begin{aligned} &REAL_{\mathcal{C}^{\mathcal{U},t,\kappa}}(m_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3) \\ &\equiv SIM_{\mathcal{C}^{\mathcal{U},t,\kappa}}(x_{\mathcal{C}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3) \end{aligned} \tag{26}$$

where $\equiv$ *indicates that the distribution is the same.*

Proof of Theorem 2. Since the server is honest, the combined view of the users in $\mathcal{C}$ is independent of the inputs of the users not in $\mathcal{C}$. SIM can simulate the views of the users in $\mathcal{C}$ perfectly by letting the users in $\mathcal{C}$ use real inputs and the others use fake inputs; e.g., the honest user chooses uniform and appropriately sized random values as inputs. Thus, the combined view of the users in $\mathcal{C}$ in SIM is the same as that in $REAL_{\mathcal{C}^{\mathcal{U},t,\kappa}}$. □

Theorem 3. *(Security against semi-honest user–server collusion) There exists a PPT simulator SIM for all* $\mathcal{U}, t, k, m_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3$*, where* $\mathcal{U} \cup S \supseteq \mathcal{C}, |\mathcal{C} \backslash S| < t, \mathcal{U} \supseteq \mathcal{U}_1 \supseteq \mathcal{U}_2 \supseteq \mathcal{U}_3$*. The output of SIM is computationally indistinguishable from that of* $REAL_{\mathcal{C}^{\mathcal{U},t,\kappa}}$*:*

$$\begin{aligned} &REAL_{\mathcal{C}^{\mathcal{U},t,\kappa}}(m_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3) \\ &\equiv SIM_{\mathcal{C}^{\mathcal{U},t,\kappa}}(m_{\mathcal{C}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3) \end{aligned} \tag{27}$$

where $\equiv$ *indicates that the distribution is the same.*

Proof of Theorem 3. A standard hybrid argument is used to prove the above theorem. The simulator SIM is constructed by performing a series of sequential modifications on the random variable REAL, and according to the transitivity of indistinguishability, if any two subsequent random variables are computationally indistinguishable, the output distributions of the simulators SIM and REAL are also computationally indistinguishable.

H_0: In this hybrid, the distribution of the combined views of the parties in SIM in $\mathcal{C}$ is the same as that in REAL.

H_1: In this hybrid, SIM replaces the ciphertext sent by the honest user to the other parties with a ciphertext that encrypts a uniformly random value (e.g., an equal-length ciphertext padded with zeros) rather than the ciphertext obtained by encrypting $s_{i,j}$. Since only the ciphertext is changed, the IND-CPA security of the symmetric encryption algorithm guarantees that the hybrid is indistinguishable from the previous hybrid.

H_2: In this hybrid, SIM lets the honest user u_i in $\mathcal{U}_2 \backslash \mathcal{C}$ replace the secret share of sk_i with a random value of uniformly suited size and then send it to the other users. The number of secret shares about the honest user sk_i in the combined view of the parties in $\mathcal{C}$ will not exceed $|\mathcal{C} \backslash S| < t$. According to the security properties of Shamir's secret sharing scheme, the adversary cannot learn each user's sk_i, which guarantees that the hybrid is indistinguishable from the previous hybrid.

H_3: In this hybrid, SIM lets each honest user u_i encrypt a random value of a uniformly suitable size instead of a_n^i and then send the ciphertext to the server. Since only the content of the ciphertext is changed, the IND-CPA security of the EC-ElGamal encryption guarantees that the hybrid is indistinguishable from the previous one.

H_4: In this hybrid, instead of using HPRG to generate the mask $\boldsymbol{r_i}$, SIM randomly selects a random value $\boldsymbol{r_i}'$ of a uniformly suited size to replace $\boldsymbol{r_i}$. Based on DDH security assumptions, HPRG guarantees that this hybrid is indistinguishable from the previous one.

H_5: In this hybrid, SIM lets each honest user replace the masked model $\widetilde{m}_i$ with a random value of a uniformly suited size. Since in the previous hybrid, $\boldsymbol{r_i}$ is chosen uniformly randomly and used as a one-time key to mask $\boldsymbol{m_i}$, $\widetilde{\boldsymbol{m}}_{\boldsymbol{i}}$ is also uniformly randomly chosen. SIM simulates REAL without knowing anything about $\boldsymbol{m_i}$, so this hybrid has the same distribution as the previous one.

H_6: In this hybrid, SIM lets each honest user u_i in $\mathcal{U}_1 \backslash \mathcal{C}$ choose a random value of uniformly suited size to replace C'_{1,u_i}, C'_{2,u_i}, and ECDLP guarantees that the hybrid is indistinguishable from the previous one.

H_7: In this hybrid, SIM lets each honest user u_i in $\mathcal{U}_1 \backslash \mathcal{C}$ send $\boldsymbol{m_i}' + \boldsymbol{r_i}'$ to the server instead of $\boldsymbol{m_i} + \boldsymbol{r_i}$, where $\boldsymbol{m_i}'$ is selected by uniformly random sampling, subject to:

$$\sum_{u_i \in \mathcal{U}_1 \backslash \mathcal{C}} \boldsymbol{m_i}' = \sum_{u_i \in \mathcal{U}_1 \backslash \mathcal{C}} \boldsymbol{m_i}$$

The distribution of $\boldsymbol{m_i} + \boldsymbol{r_i}$ is the same as that of $\boldsymbol{m_i}' + \boldsymbol{r_i}'$, which is subject to the above equation.

This paper defines a PPT simulator SIM based on the last hybrid. The combined view of the parties in $\mathcal{C}$ in SIM is computationally indistinguishable from the real execution of REAL; thus, the proof is completed. □

6. Performance Evaluation

6.1. Complexity Analysis

In this section, the performance of the proposed scheme is analyzed from computational complexity, communication complexity, and user dropout, and it is compared with other schemes. The results are shown in Table 3.

Table 3. Comparison of communication and computational complexity with other schemes.

		SecAgg	TurboAgg	SecAgg++	FastSecAgg	Paper [41]	Ours
computation complexity	User	$O(Ln+n^2)$	$O(Llognlog^2logn)$	$O(Llogn+log^2n)$	$O(Llogn)$	$O(n^2+L)$	$O(L)$
	Server	$O(Ln^2)$	$O(Llognlog^2logn)$	$O(Lnlogn+nlog^2n)$	$O(Llogn)$	$O(n)$	$O(n+L)$
communication complexity	User	$O(L+n)$	$O(Llogn)$	$O(L+logn)$	$O(L+n)$	$0(L+n)$	$O(L)$
	Server	$O(nL+n^2)$	$O(Lnlogn)$	$O(nL+nlogn)$	$O(nL+n^2)$	$O(nL+n^2)$	$O(nL+n^2)$
communication rounds		4	$n/logn$	3	3	3	2–3

Computation complexity: User computation overhead comes mainly from masking the local model using masks after training is completed, and the computational complexity is $O(L)$, where L denotes the size of the local model $\boldsymbol{m_i}$. Compared to the studies [24,41], which utilized secret sharing to share seeds, our scheme reduces the total complexity from $O(n^2+L)$ to $O(L)$. The computational overhead of the server mainly comes from: (1) computing $C_{1,s}, C_{2,s}$ after receiving n pairs of seed ciphertexts, with a computational complexity of $O(n)$; (2) computing $C'_{1,s}$ and $C'_{2,s}$, with a computational complexity of $O(n)$; (3) reconstructing the sum of sk_i of the dropped out user using n secret shares, with a computational complexity of $O(n)$ (here, this paper adopts the method in [24] to reconstruct the secret, so the computational complexity is $O(n)$, instead of $O(n^2)$ in Shamir's scheme); and (4) calculating $\boldsymbol{R}$ according to HPRG, the computational complexity of which is $O(L)$. Thus, the total complexity is $O(L+n)$.

Communication complexity: User communication overhead comes mainly from: (1) sending the masked model $\widetilde{m}_i$ containing L elements with a communication complexity of $O(L)$; (2) sending the seed ciphertexts $C_{1,u_i}, C_{2,u_i}, C'_{1,u_i}, C'_{2,u_i}$ and receiving $C_{1,s}, C_{2,s}$, with a communication complexity of $O(1)$; and (3) sending the sum of secret shares, with a total communication complexity of $O(L)$. Compared to SecAgg, our scheme reduces the overhead of $2n$ key agreements and $2n$ secret shares. The communication overhead of the server mainly comes from: (1) receiving the masked local models submitted by n users, with a communication complexity of $O(nL)$; and(2) receiving the (C_{1,u_i}, C_{2,u_i}) submitted by n users and forwarding the computation results to n users, with a communication complexity of $O(n^2)$. Therefore, the total communication complexity is $O(nL+n^2)$.

User dropout: In Section 4.3.3, to make the aggregation scheme applicable to cross-device FL, this paper increases the tolerance to user dropouts by introducing secret sharing before the training begins, and by exploiting the additive homomorphism of secret sharing, the server only requires a single reconfiguration to eliminate the impact of the dropped out user, and the aggregation time does not increase with the number of dropped out users. When no user drops out, the user and the server can complete the aggregation with only two rounds of communication, and once a user drops out, an additional round of communication is needed to assist the server in eliminating the impact of the dropped out user. Appendices A and B demonstrate that this is because the PK used in this aggregation contains the public key of the dropped out user. Therefore, to avoid the influence of the

dropped out user on the subsequent aggregation, the server recalculates the PK at the end of the previous round of aggregation using the public key of the surviving user.

6.2. Experimental Setup

The prototype is implemented using Python 3.7.11 and some of its standard libraries such as Gmpy2, Cryptography, etc., and it executes on a computer equipped with Intel(R) Xeon(R) Gold 5218 CPU @ 2.30 GHz and 256 GB of RAM while running on Ubuntu 18.04 operating system. Sockets are used for the communication between the user and the server. EC-ElGamal is implemented using NIST P-256 elliptic curves; elliptic curve Diffie–Hellman and SHA-256 are used for key agreement; AES-GCM encrypts the secret share using 256-bit keys; standard Shamir secret share is used for secret sharing using; HPRGs are constructed based on SHA-256; seed size is 98-bit; and the number of congruent cosets in CRT is 10.

6.3. Experimental Results

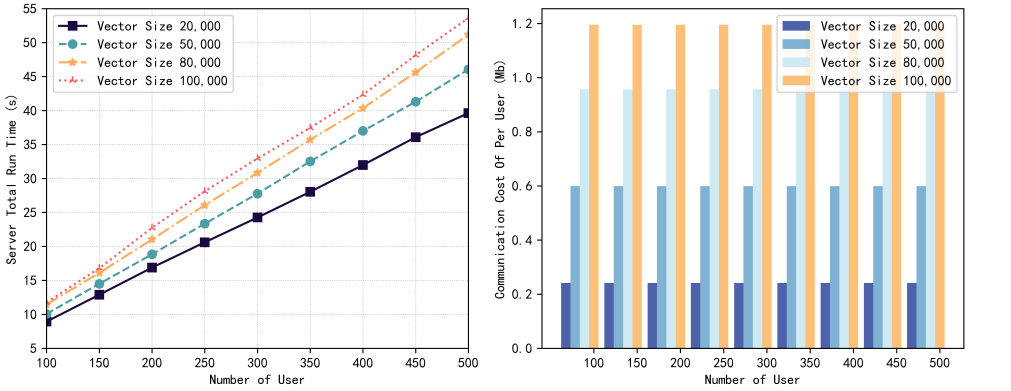
Theorem 1 proves that the global model is equal to the sum of all user's local models; therefore, without considering the loss of accuracy of local models converted from floating-point to integer, the scheme proposed in this paper basically does not compromise the performance of the global model. The experiments will mainly focus on the privacy-preserving aggregation of local models. In cross-device FL, the clients are mostly resource-limited devices, and the computation and communication overheads of the aggregation scheme are very critical. Furthermore, due to the network state and energy issues, it is common for users to drop out during the training process, so the computation and communication overheads associated with dealing with dropped out users are also important to focus on. This paper skips the user's local training process, uses vectors of different sizes to replace the user's local model, considers the server sending the global model to the user as the beginning of the aggregation and the server aggregating a new global model as the end, and then evaluates the efficiency of our scheme based on the computation and communication overheads during the aggregation process. In addition, we further evaluate the performance of the scheme by comparing it with the SecAgg scheme. We did not consider other aggregation schemes [25,35,38] because they would either reduce security guarantees or dropout tolerance while improving aggregation efficiency.

First, this paper evaluates the effect of the number of users and the vector size on the total server runtime (including wait time) and the communication overhead of the users when there are no user dropouts through experimentation.

The results are illustrated in Figures 3 and 4, and it can be observed that the server runtime increases almost linearly with the number of users and the vector size. Particularly, compared to the vector size, the number of users has slightly more effect on the server runtime. Meanwhile, the change in the number of users has almost no effect on the communication overhead of users, which is mainly affected by the input vector size, and this is consistent with our theoretical analysis. When the number of users is 500 and the vector size is 100 K, the whole aggregation process can be completed in about 55 s, and the communication cost of each user is only about 1.2 MB, which proves the efficiency of the scheme proposed in this paper.

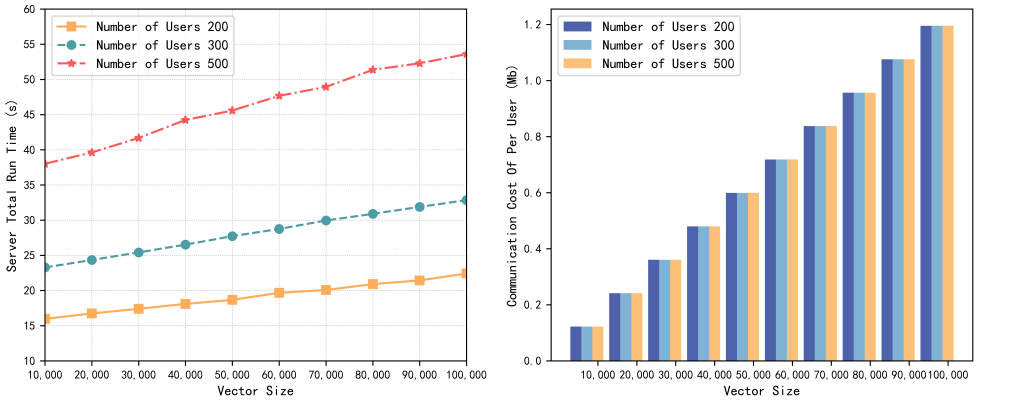
Next, this paper considers the effect of user dropouts on aggregation efficiency. First, the impact brought by the change in the number of users on the server running time and user communication overhead under different dropout rates is investigated, and the experimental results are shown in Figure 5a,b. It can be seen that the total server runtime in SecAgg grows exponentially with the number of users under different dropout rates, and the scheme proposed in this paper grows almost linearly. The advantages of our schemes gradually come to the fore as the number of users and the dropout rate increase. When the dropout rate is 0.3 and there are 500 users, the server runtime in SecAgg is about 24 times longer than that of our scheme. For communication overhead, as the number of users increases, the communication overhead of users in SecAgg grows linearly, and the

larger the dropout rate is, the faster it grows, while the communication overhead of users in our scheme is almost unaffected by the number of users and the dropout rate.



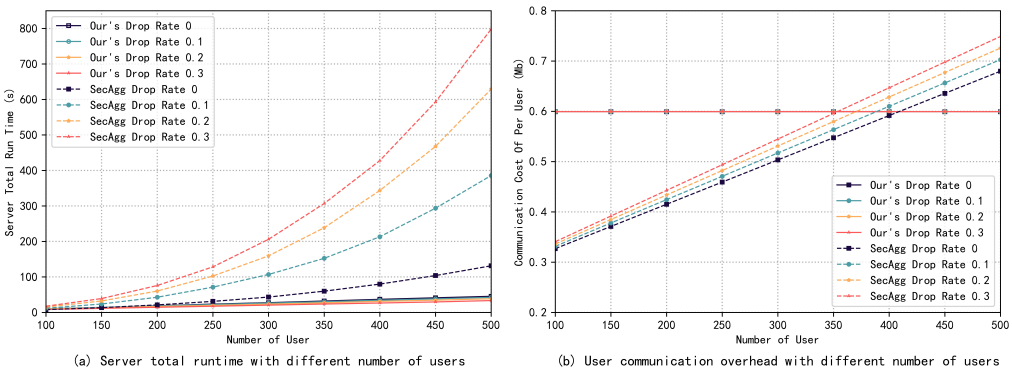
(a) Server total runtime with different number of users (b) User communication overhead with different number of users

Figure 3. Server runtime and user communication overhead with different numbers of users. No users dropped out.



(a) Server total runtime with different vector sizes (b) User communication overhead with different vector sizes

Figure 4. Server runtime and user communication overhead with different vector sizes. No users dropped out.



(a) Server total runtime with different number of users (b) User communication overhead with different number of users

Figure 5. Server runtime and user communication overhead with different numbers of users. Dropout rate constant.

Then, this paper compares the effect of different dropout rates on the total server runtime and the user communication overhead under a fixed number of users, and the experimental results are presented in Figure 6a,b. It can be seen that as the dropout rate increases, the server runtime in SecAgg increases significantly, while the server runtime in our scheme decreases. This is because when a user drops out, the integer space that

the server needs to search when decrypting the same coset ciphertext is smaller, and the computation time decreases. Meanwhile, the communication overhead of users in SecAgg increases linearly with the increase in the dropout rate because they need to send more secret shares to eliminate the effect of the dropped out users. In contrast, the user communication overhead in our scheme is almost independent of the dropout rate, and owing to the homomorphism of the secret share, the user only needs to send one secret share in the Unmask Model phase. Experimental results indicate that our scheme has a stronger dropout tolerance and is more applicable in cross-device FL.

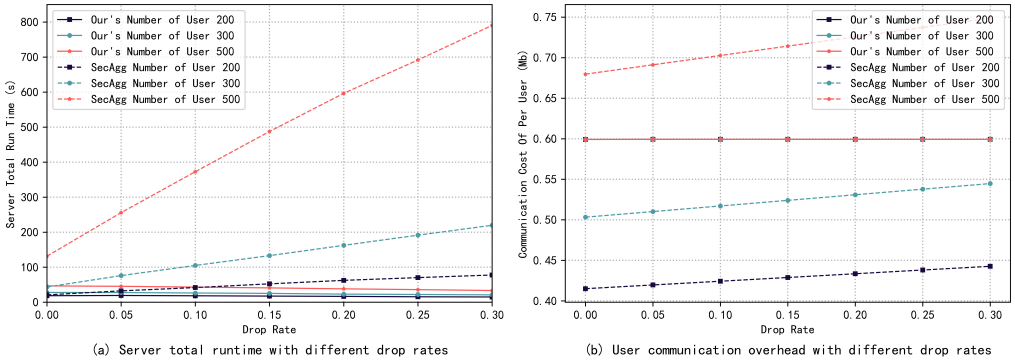
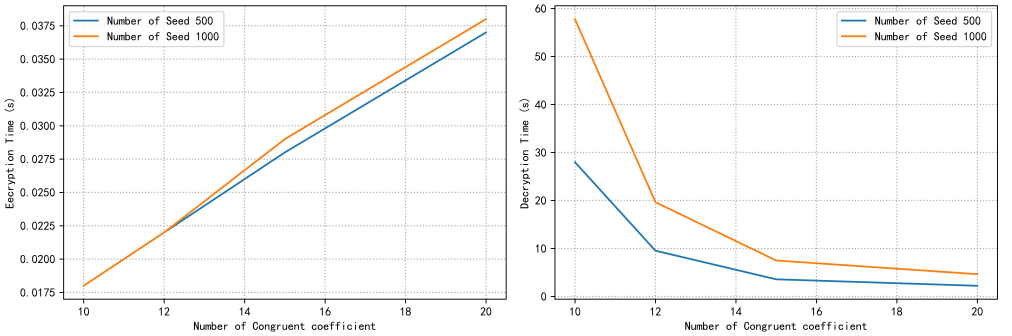


Figure 6. Server runtime and user communication overhead with different dropout rates. Number of users is constant.

Table 4 lists the running time of users and servers in different phases of aggregation under different dropout rates and the total time. It can be observed that most of the time in the aggregation process is consumed in the final mask removal phase. This is because the server needs to solve the ECDLP, and the search space gradually increases with the number of users, and the computation time will increase gradually. The computation time can be reduced by reasonably increasing the number of congruent coefficients. As shown in Figure 7, adding a few congruent coefficients significantly reduces the decryption time and thus significantly improves the aggregation efficiency. The increased encryption time for the user is very slight. In addition, CTR and baby-step giant-step (BSGS) algorithms can be combined to further improve decryption efficiency [46], and in practical applications, aggregation efficiency can also be enhanced by high-performance servers and multi-threaded parallel computing.

Table 4. Runtime and total time for different phases of users and servers with different dropout rates. Vector size: 50 K.

	Number of User	Dropout Rate	Collecting Masked Model	Key Conversion (Unmask Model)	Unmask Model	Total Time
User	500	0%	8.62 s	0.18 s	0 s	8.8 s
User	500	10%	7.79 s	2.42 s	0.05 s	10.26 s
User	500	20%	7.05 s	2.26 s	0.08 s	9.39 s
User	500	30%	5.96 s	2.16 s	0.10 s	8.22 s
Server	500	0%	9.89 s	35.46 s	0 s	45.35 s
Server	500	10%	8.00 s	2.36 s	32.11 s	42.47 s
Server	500	20%	6.83 s	2.22 s	29.41 s	38.46 s
Server	500	30%	6.03 s	2.10 s	25.23 s	33.36 s



(a) Encryption time with different numbers of congruent coefficient (b) Decryption time with different number of congruent coefficient

Figure 7. Encryption and decryption times with different numbers of congruent coefficient.

7. Conclusions

In this paper, an efficient FL secure aggregation method is proposed based on multi-homomorphic attributes. The proposed method allows users to protect the privacy of the local model with $O(L)$ computational and communication overheads, without affecting the server aggregating the global model while tolerating user dropouts at any time. The correctness, security, and performance of our scheme are analyzed and evaluated experimentally. The experimental evaluation results indicate that our scheme has high practicality and efficiency, and it significantly improves aggregation efficiency and has a stronger tolerance to user dropouts compared to other secure aggregation schemes. In future work, we will consider how to achieve the secure aggregation of models when participants are malicious.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Q.G., Y.S. and X.C.; data curation, Y.W.; formal analysis, Q.G., Y.S. and X.C.; methodology, Q.G., Y.S. and F.Y.; writing—original draft, Q.G.; writing—review and editing, Q.G. and Y.S. All authors reviewed the results and approved the final version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The relevant data are available within the paper.

Acknowledgments: We express our gratitude to the relevant personnel from Information Engineering University who largely helped us with our scheme.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Based on C'_{1,u_i}, C'_{2,u_i} sent by each user, the server computes:

$$C'_{1,s} = \sum_{u_i \in \mathcal{U}} k_i G \tag{A1}$$

$$\begin{aligned} C'_{2,s} &= C_{2,s} + \sum_{u_i \in \mathcal{U}} C'_{2,u_i} \\ &= \sum_{u_i \in \mathcal{U}} v_i G + \sum_{u_i \in \mathcal{U}} k_i PK + \sum_{u_i \in \mathcal{U}} (-C_{1,s} sk_i + k_i pk_s) \\ &= \sum_{u_i \in \mathcal{U}} v_i G + \sum_{u_i \in \mathcal{U}} k_i PK - \sum_{u_i \in \mathcal{U}} \sum_{u_i \in \mathcal{U}} C_{1,s} sk_i + \sum_{u_i \in \mathcal{U}} k_i pk_s \\ &= \sum_{u_i \in \mathcal{U}} v_i G + \sum_{u_i \in \mathcal{U}} k_i PK - \sum_{u_i \in \mathcal{U}} \sum_{u_i \in \mathcal{U}} k_i G \sum_{u_i \in \mathcal{U}} sk_i + \sum_{u_i \in \mathcal{U}} k_i pk_s \end{aligned} \tag{A2}$$

According to Equation (1), $pk_i = Gsk_i$:

$$\begin{aligned} C'_{2,s} &= \sum_{u_i \in \mathcal{U}} v_i G + \sum_{u_i \in \mathcal{U}} k_i PK - \sum_{u_i \in \mathcal{U}} \sum_{u_i \in \mathcal{U}} k_i G \sum_{u_i \in \mathcal{U}} sk_i + \sum_{u=0} k_i pk_s \\ &= \sum_{u_i \in \mathcal{U}} v_i G + \sum_{u_i \in \mathcal{U}} k_i PK - \sum_{u_i \in \mathcal{U}} \sum_{u_i \in \mathcal{U}} k_i pk_i + \sum_{u_i \in \mathcal{U}} k_i pk_s \end{aligned} \tag{A3}$$

According to $PK = \sum_{u_i \in \mathcal{U}} pk_i$:

$$\begin{aligned} C'_{2,s} &= \sum_{u_i \in \mathcal{U}} v_i G + \sum_{u_i \in \mathcal{U}} k_i \sum_{u_i \in \mathcal{U}} pk_i - \sum_{u_i \in \mathcal{U}} \sum_{u_i \in \mathcal{U}} k_i pk_i \\ &+ \sum_{u_i \in \mathcal{U}} k_i pk_s = \sum_{u_i \in \mathcal{U}} v_i G + \sum_{u_i \in \mathcal{U}} k_i pk_s \end{aligned} \tag{A4}$$

In summary:

$$\begin{aligned} C'_{1,s} &= \sum_{u_i \in \mathcal{U}} k_i G \\ C'_{2,s} &= \sum_{u_i \in \mathcal{U}} v_i G + \sum_{u_i \in \mathcal{U}} k_i pk_s \end{aligned} \tag{A5}$$

Appendix B

Due to the existence of dropout users, the server computes $C'_{2,s}$ according to Equation (14):

$$C'_{1,s} = \sum_{u_i \in \mathcal{U}_2} k_i G \tag{A6}$$

$$\begin{aligned} C'_{2,s} &= C_{2,s} + \sum_{u_i \in \mathcal{U}_2} C'_{2,u_i} \\ &= \sum_{u_i \in \mathcal{U}_1} v_i G + \sum_{u_i \in \mathcal{U}_1} k_i PK + \sum_{u_i \in \mathcal{U}_2} (-C_{1,s} sk_i + k_i pk_s) \\ &= \sum_{u_i \in \mathcal{U}_1} v_i G + \sum_{u_i \in \mathcal{U}_1} k_i PK \\ &- C_{1,s} \sum_{u_i \in \mathcal{U}_2} sk_i + \sum_{u_i \in \mathcal{U}_2} k_i pk_s \\ &= \sum_{u_i \in \mathcal{U}_1} v_i G + \sum_{u_i \in \mathcal{U}_1} k_i \sum_{u_i \in \mathcal{U}} pk_i \\ &- \sum_{u_i \in \mathcal{U}_1} k_i G \sum_{u_i \in \mathcal{U}_2} sk_i + \sum_{u_i \in \mathcal{U}_2} k_i pk_s \\ &= \sum_{u_i \in \mathcal{U}_1} v_i G + \sum_{u_i \in \mathcal{U}_1} k_i \sum_{u_i \in \mathcal{U}} sk_i G \\ &- \sum_{u_i \in \mathcal{U}_1} k_i \sum_{u_i \in \mathcal{U}_2} sk_i G + \sum_{u_i \in \mathcal{U}_2} k_i pk_s \\ &= \sum_{u_i \in \mathcal{U}_1} v_i G + \sum_{u_i \in \mathcal{U}_1} k_i (\sum_{u_i \in \mathcal{U}} sk_i G - \sum_{u_i \in \mathcal{U}_2} sk_i G) + \sum_{u_i \in \mathcal{U}_2} k_i pk_s \\ &= \sum_{u_i \in \mathcal{U}_1} v_i G + \sum_{u_i \in \mathcal{U}_1} k_i (G \sum_{u_i \in \mathcal{U}} sk_i - \sum_{u_i \in \mathcal{U}_2} sk_i) + \sum_{u_i \in \mathcal{U}_2} k_i pk_s \\ &= \sum_{u_i \in \mathcal{U}_1} v_i G + \sum_{u_i \in \mathcal{U}_1} k_i (G \sum_{u_i \in \mathcal{U} \setminus \mathcal{U}_2} sk_i) + \sum_{u_i \in \mathcal{U}_2} k_i pk_s \\ &= \sum_{u_i \in \mathcal{U}_1} v_i G + \sum_{u_i \in \mathcal{U}_1} k_i G (\sum_{u_i \in \mathcal{U} \setminus \mathcal{U}_2} sk_i) + \sum_{u_i \in \mathcal{U}_2} k_i pk_s \\ &= \sum_{u_i \in \mathcal{U}_1} v_i G + C_{1,s} \sum_{u_i \in \mathcal{U} \setminus \mathcal{U}_2} sk_i + \sum_{u_i \in \mathcal{U}_2} k_i pk_s \end{aligned} \tag{A7}$$

As we can see, due to the dropped out user, $C'_{2,s}$ is not of the form $P_m G + KQ$, where $Q = pk_s$, $K = \sum_{u_i \in \mathcal{U}_2} k_i$, and therefore cannot be decrypted using sk_s and $C'_{1,s}$. To com-

plete the key conversion, $C_{1,s}\sum_{u_i\in\mathcal{U}\setminus\mathcal{U}_2} sk_i$ needs to be removed. Therefore, we define the following equation:

$$C'_{2,s} = C_{2,s} + \sum_{u_i\in\mathcal{U}_2} C'_{2,u_i} + C_{1,s}\sum_{u_i\in\mathcal{U}\setminus\mathcal{U}_2}(-sk_i) \tag{A8}$$

Then, the key conversion is achieved by reconstructing the $sk_i, u_i \in \mathcal{U}\setminus\mathcal{U}_2$ of the dropped user.

References

1. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 2017 International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 20–22 April 2017; Aarti, S., Jerry, Z., Eds.; Proceedings of Machine Learning Research: Brookline, MA, USA, 2017; Volume 54, pp. 1273–1282.
2. Li, L.; Fan, Y.X.; Tse, M.; Lin, K.Y. A review of applications in federated learning. *Comput. Ind. Eng.* **2020**, *149*, 106854. [CrossRef]
3. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Process. Mag.* **2020**, *37*, 50–60. [CrossRef]
4. Imteaj, A.; Thakker, U.; Wang, S.; Li, J.; Amini, M.H. A Survey on Federated Learning for Resource-Constrained IoT Devices. *IEEE Internet Things J.* **2022**, *9*, 1–24. [CrossRef]
5. Hard, A.; Rao, K.; Mathews, R.; Beaufays, F.; Augenstein, S.; Eichner, H.; Kiddon, C.; Ramage, D. Federated Learning for Mobile Keyboard Prediction. *arXiv* **2018**, arXiv:1811.03604.
6. Nasr, M.; Shokri, R.; Houmansadr, A. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In Proceedings of the 40th IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 739–753. [CrossRef]
7. Shokri, R.; Stronati, M.; Song, C.Z.; Shmatikov, V. Membership Inference Attacks Against Machine Learning Models. In Proceedings of the 2017 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 22–26 May 2017; pp. 3–18. [CrossRef]
8. Melis, L.; Song, C.; Cristofaro, E.D.; Shmatikov, V. Exploiting Unintended Feature Leakage in Collaborative Learning. In Proceedings of the 40th IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 691–706. [CrossRef]
9. Dong, Y.; Hou, W.; Chen, X.; Zeng, S. Efficient and Secure Federated Learning Based on Secret Sharing and Gradients Selection. *J. Comput. Res. Dev.* **2020**, *57*, 2241–2250.
10. Phong, L.T.; Aono, Y.; Hayashi, T.; Wang, L.; Moriai, S. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 1333–1345. [CrossRef]
11. Mansouri, M.; Önen, M.; Jaballah, W.B.; Conti, M. SoK: Secure Aggregation Based on Cryptographic Schemes for Federated Learning. *Proc. Priv. Enhancing Technol.* **2023**, *2023*, 140–157. [CrossRef]
12. Liu, Z.; Guo, J.; Yang, W.; Fan, J.; Lam, K.Y.; Zhao, J. Privacy-Preserving Aggregation in Federated Learning: A Survey. *IEEE Trans. Big Data* **2022**, 1–20. [CrossRef]
13. Cynthia, D.; Aaron, R. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* **2014**, *9*, 407. [CrossRef]
14. 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. [CrossRef]
15. Kim, M.; Gunlu, O.; Schaefer, R.F. Federated Learning with Local Differential Privacy: Trade-Offs Between Privacy, Utility, and Communication. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 6–11 June 2021; pp. 2650–2654. [CrossRef]
16. Shayan, M.; Fung, C.; Yoon, C.J.M.; Beschastnikh, I. Biscotti: A Blockchain System for Private and Secure Federated Learning. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 1513–1525. [CrossRef]
17. So, J.; Güler, B.; Avestimehr, A.S. Byzantine-Resilient Secure Federated Learning. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 2168–2181. [CrossRef]
18. Sotthiwat, E.; Zhen, L.; Li, Z.; Zhang, C. Partially Encrypted Multi-Party Computation for Federated Learning. In Proceedings of the 21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Melbourne, Australia, 10–13 May 2021; pp. 828–835. [CrossRef]
19. Liu, C.; Chakraborty, S.; Verma, D. Secure Model Fusion for Distributed Learning Using Partial Homomorphic Encryption. In *Policy-Based Autonomic Data Governance*; Springer: Cham, Switzerland, 2019; pp. 154–179. [CrossRef]
20. Zhang, C.; Li, S.; Xia, J.; Wang, W.; Yan, F.; Liu, Y. BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning. In Proceedings of the USENIX Annual Technical Conference, Berkeley, CA, USA, 15–17 July 2020; pp. 493–506.
21. Kairouz, P.; McMahan, H.B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A.N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al. Advances and Open Problems in Federated Learning. *Found. Trends Mach. Learn.* **2021**, *14*, 1–210. [CrossRef]

22. Lyu, L.; Bezdek, J.C.; He, X.; Jin, J. Fog-Embedded Deep Learning for the Internet of Things. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4206–4215. [CrossRef]
23. Zhou, C.Y.; Fu, A.M.; Yu, S.; Yang, W.; Wang, H.Q.; Zhang, Y.Q. Privacy-Preserving Federated Learning in Fog Computing. *IEEE Internet Things J.* **2020**, *7*, 10782–10793. [CrossRef]
24. 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. [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. Bell, J.H.; Bonawitz, K.A.; Gascón, A.; Lepoint, T.; Raykova, M. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (ACM CCS), Virtual Event, 9–13 November 2020; pp. 1253–1269. [CrossRef]
27. Choi, B.; Sohn, J.y.; Han, D.J.; Moon, J. Communication-Computation Efficient Secure Aggregation for Federated Learning. *arXiv* **2020**, arXiv:2012.05433.
28. Jiang, Z.; Wang, W.; Liu, Y. FLASHE: Additively Symmetric Homomorphic Encryption for Cross-Silo Federated Learning. *arXiv* **2021**, arXiv:2109.00675.
29. Mandal, K.; Gong, G. NIKE-based Fast Privacy-preserving High-dimensional Data Aggregation for Mobile Devices. *IEEE Trans. Dependable Secur. Comput.* **2018**, *27*, 142–149.
30. Ma, J.; Naas, S.; Sigg, S.; Lyu, X. Privacy-preserving federated learning based on multi-key homomorphic encryption. *Int. J. Intell. Syst.* **2022**, *37*, 5880–5901. [CrossRef]
31. 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**, *9*, 4923–4937. [CrossRef]
32. Xu, Y.; Peng, C.; Tan, W.; Tian, Y.; Ma, M.; Niu, K. Non-interactive verifiable privacy-preserving federated learning. *Future Gener. Comput. Syst.* **2022**, *128*, 365–380. [CrossRef]
33. Liu, R.; Cao, Y.; Yoshikawa, M.; Chen, H. FedSel: Federated SGD Under Local Differential Privacy with Top-k Dimension Selection. In Proceedings of the 25th International Conference on Database Systems for Advanced Applications (DASFAA), Jeju, Republic of Korea, 24–27 September 2020; Nah, Y., Cui, B., Lee, S.W., Yu, J.X., Moon, Y.S., Whang, S.E., Eds.; Springer: Cham, Switzerland, 2020; Volume 12112, pp. 485–501. [CrossRef]
34. Wei, K.; Li, J.; Ding, M.; Ma, C.; Yang, H.H.; Farokhi, F.; Jin, S.; Quek, T.Q.S.; Poor, H.V. Federated Learning With Differential Privacy: Algorithms and Performance Analysis. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 3454–3469. [CrossRef]
35. 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]
36. Jahani-Nezhad, T.; Maddah-Ali, M.A.; Li, S.; Caire, G. SwiftAgg: Communication-Efficient and Dropout-Resistant Secure Aggregation for Federated Learning with Worst-Case Security Guarantees. In Proceedings of the 2022 IEEE International Symposium on Information Theory (ISIT), Espoo, Finland, 26 June–1 July 2022; pp. 103–108. [CrossRef]
37. Kadhe, S.; Rajaraman, N.; Koyluoglu, O.O.; Ramchandran, K. FastSecAgg: Scalable Secure Aggregation for Privacy-Preserving Federated Learning. *arXiv* **2020**, arXiv:2009.11248.
38. Zheng, Y.; Lai, S.; Liu, Y.; Yuan, X.; Yi, X.; Wang, C. Aggregation Service for Federated Learning: An Efficient, Secure, and More Resilient Realization. *IEEE Trans. Dependable Secur. Comput.* **2022**, *20*, 988–1001. [CrossRef]
39. Nasirigerdeh, R.; Torkzadehmahani, R.; Matschinske, J.; Baumbach, J.; Rueckert, D.; Kaissis, G. HyFed: A Hybrid Federated Framework for Privacy-preserving Machine Learning. *arXiv* **2021**, arXiv:2105.10545.
40. Zhao, Y.; Sun, H. Information Theoretic Secure Aggregation with User Dropouts. *IEEE Trans. Inf. Theory* **2022**, *68*, 7471–7484. [CrossRef]
41. Liu, Z.; Guo, J.; Lam, K.Y.; Zhao, J. Efficient Dropout-resilient Aggregation for Privacy-preserving Machine Learning. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 1839–1854. [CrossRef]
42. Li, Y.; Zhou, Y.; Jolfaei, A.; Yu, D.; Xu, G.; Zheng, X. Privacy-Preserving Federated Learning Framework Based on Chained Secure Multiparty Computing. *IEEE Internet Things J.* **2021**, *8*, 6178–6186. [CrossRef]
43. Koblitz, N. Elliptic curve cryptosystems. *Math. Comput.* **1987**, *48*, 203–209. [CrossRef]
44. Elgamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* **1985**, *31*, 469–472. [CrossRef]
45. Hu, Y.; Martin, W.J.; Sunar, B. Enhanced Flexibility for Homomorphic Encryption Schemes via CRT. In Proceedings of the Applied Cryptography and Network Security, Singapore, 26–29 June 2012.
46. Shafagh, H.; Hithnawi, A.; Burkhalter, L.; Fischli, P.; Duquennoy, S. Secure Sharing of Partially Homomorphic Encrypted IoT Data. In Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems (SenSys), Delft, The Netherlands, 6–8 November 2017; pp. 1–14. [CrossRef]
47. Boneh, D.; Lewi, K.; Montgomery, H.; Raghunathan, A. Key Homomorphic PRFs and Their Applications. In *Advances in Cryptology, Proceedings of the CRYPTO 2013, Santa Barbara, CA, USA, 18–22 August 2013*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2013; pp. 410–428. [CrossRef]

48. Naor, M.; Pinkas, B.; Reingold, O. Distributed Pseudo-random Functions and KDCs. In *Advances in Cryptology, Proceedings of the EUROCRYPT '99, Prague, Czech Republic, 2–6 May 1999*; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1592, pp. 327–346.
49. Everspaugh, A.; Paterson, K.G.; Ristenpart, T.; Scott, S. Key Rotation for Authenticated Encryption. In Proceedings of the 37th Annual International Cryptology Conference (Crypto), Santa Barbara, CA, USA, 20–24 August 2017; Volume 10403, pp. 98–129. [CrossRef]
50. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613. [CrossRef]
51. Sklavos, N. Cryptography and Network Security: Principles and Practice. *Inf. Secur. J. Glob. Perspect.* **2013**, *23*, 49–50. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.