electronics

Article

Cluster-Based Secure Aggregation for Federated Learning

Jien Kim, Gunryeong Park, Miseung Kim and Soyoung Park *

Department of Computer Science and Engineering, Konkuk University, Seoul 05025, Republic of Korea
* Correspondence: soyoungpark@konkuk.ac.kr; Tel.: +82-2-450-0482

Abstract: In order to protect each node's local learning parameters from model inversion attacks, secure aggregation has become the essential technique for federated learning so that the federated learning server knows only the combined result of all local parameters. In this paper, we introduced a novel cluster-based secure aggregation model that effectively deals with dropout nodes while reducing communicational and computational overheads. Specifically, we considered a federated learning environment with heterogeneous devices deployed across the country. The computing power of each node and the amount of training data can be heterogeneous. Because of this, each node had a different local processing time, and the response time to the server is also different. To clearly determine the dropout nodes in this environment, our model clusters nodes with similar response times based on each node's local processing time and location and then performs the aggregation on a pre-cluster basis. In addition, we propose a new practical additive sharing-based masking protocol to hide the actual local updates of nodes during aggregation. The new masking protocol makes it easy to remove the share of dropout nodes from the aggregation without using a (t, n) threshold scheme, and updates from dropout nodes are still secure even if they are delivered to the server after the dropout shares have been revealed. In addition, our model provides mask verification for reliable aggregation. Nodes can publicly verify the correctness and integrity of the masks received from others using a discrete logarithm problem before the aggregation. As a result, the proposed aggregation model is robust to dropout nodes and ensures the privacy of local updates if at least three honest nodes are alive in each cluster. Since the masking process is performed on a cluster basis, our model effectively reduces the overhead of generating and sharing the masking value. For an average cluster size $\mathcal{C}$ and a total number of nodes N, the computation and communication cost of each node is $O(\mathcal{C})$, the computation cost of the server is $O(N)$, and the communication cost is $O(N\mathcal{C})$. We analyzed the security and efficiency of our protocol by simulating diverse dropout scenarios. The simulated results showed that our cluster-based secure aggregation outputs about a 91% learning accuracy regardless of dropout rate with four clusters for one hundred nodes.

Keywords: secure aggregation; federated learning; clustering; pairwise mask; additive sharing; dropout; efficiency

Citation: Kim, J.; Park, G.; Kim, M.; Park, S. Cluster-Based Secure Aggregation for Federated Learning. *Electronics* **2023**, *12*, 870. https://doi.org/10.3390/electronics12040870

Academic Editors: Jiliang Zhang, Zhaojun Lu, He Li and Zukun Lu

Received: 6 January 2023
Revised: 3 February 2023
Accepted: 7 February 2023
Published: 8 February 2023

1. Introduction

Federated learning [1], which performs machine learning on the entire training data distributed across multiple external nodes, has a great advantage in protecting user privacy for the data used for training. Each node's training data does not need to be collected or exposed; instead, each node uses its own training data to perform the learning using only the local training result. The federated learning server continuously interacts with the node's local learning parameters until the weight parameters converge. Here, secure aggregation is necessary to hide the real learning parameters of nodes from the federated server and the other nodes because the local learning model can be reconstructed with the local parameters, which is known as a model inversion attack [2–4]. Secure aggregation allows the federated server to obtain the total sum and average of all the local parameters without knowing the actual local parameters.

Recently, many secure aggregation techniques for federated learning have been proposed. Secure aggregation protocols based on secure multiparty computation [5,6], homomorphic encryptions [7–10], and differential privacy [11,12] have been proposed, and masking techniques to protect each node's local result [13–18], methods to efficiently handle dropout nodes [7,8] and reduce communication overhead [19–22], studies to detect Byzantine users [23–25], and methods to validate aggregation results [26,27] have been conducted. Many previous studies assumed that each node had the same size training data distributed in an independent and identical distribution, as well as the same computing power. However, in practice, the computing power of nodes is not the same, and the size of training data held by each node is also different.

In this paper, we introduced a new simple and practical cluster-based secure aggregation (CSA) technique that is suitable for federated learning involving heterogeneous devices. We considered the following federated learning environments: (1) a large number of nodes participating in federated learning; (2) each node having a different computing power; (3) the amount of training data held by each node being different; and (4) nodes being widely distributed across the country. In this context, the proposed CSA model specifically solves the following two problems: the local learning result of each node potentially not being integrated at the same rate, and the response time of each node being different.

The CSA technique assigns different training weights to nodes according to the size of training data each node has and performs secure aggregation based on the weights. This solves the problem of training data size. The problem of differences in the response time may affect the overall throughput of federated learning. Federated learning works iteratively in a synchronized manner, so the server should wait for all responses from all nodes in each iteration. Assuming all nodes respond to the server, the server waits unconditionally until it receives responses from all nodes. However, dropout nodes can occur for any reason (whether it is a problem with the node itself or a communication problem). Therefore, the server cannot wait indefinitely for responses from nodes and must determine an appropriate waiting time. If the server latency is too long, the overall federated learning throughput will decrease, and if the latency is too short, there may too many false positives for dropouts. Therefore, in situations where nodes have different response times, the server can efficiently determine whether a message is dropped or a non-response is due to a communication delay, and it can also reduce the number of false positives that it leaves. To solve this problem, a cluster-based secure aggregation strategy was proposed. The strategy is simple: it clusters nodes according to the response time.

Each node's response time is determined by the node's local processing time and communication latency. The local processing time of each node is mainly affected by the computing power and the size of training data, and this can be evaluated by each node. On the other hand, communication latency depends on the network environment and communication distance. Assessing the actual network environment from each node to the server is not easy. Therefore, only the communication distance is considered to evaluate the communication delay.

We defined a processing score that represents the processing time of each node. The processing score is defined by the computational capacity of each node and the size of training data. The proposed CSA technique first clusters nodes according to processing scores and GPS information. We proposed a grid-based clustering algorithm for clustering nodes, since all the nodes in a cluster have similar processing scores and communication distances, and their response time to the server is also similar. Thus, the server can estimate the maximum waiting time for a cluster, which can then be used to determine dropout. The aggregated intermediate sums for each cluster are finally aggregated for all clusters.

In particular, we proposed a highly efficient mask-based aggregation algorithm that does not use a (t, n) threshold secret sharing scheme to reduce the computational and communication costs. Each node first generates and provides random masks to the other users in the same cluster. The validity of the given masks can be verified publicly, and then the masked updates of the nodes are aggregated. The random masks are finally removed after

the aggregation, and only the sum of local parameters can be computed. Recommended aggregation is also robust for suspended dropout users. To deal with dropout nodes, Shamir's (t, n) threshold secret sharing scheme is commonly used to reconstruct the masks of dropout users. However, the computation and communication overhead due to the use of the scheme is very large. Moreover, a verifiable secret sharing scheme may be required to verify shares. The proposed mask-based aggregation does not use a (t, n) threshold secret sharing scheme to remove the random masks of dropout users, so all the overheads caused by using the secret sharing scheme are eliminated. In Section 3, we will describe in detail how our method generates a correct aggregated result for existing users.

The main contributions of this paper are summarized as follows:

- We proposed a new cluster-based secure aggregation (CSA) strategy for federated learning with heterogeneous nodes that have different computing powers and different sizes of training data.
- The CSA technique clusters with similar response times. We introduced a processing score to represent the processing time of each node and proposed a new grid-based clustering algorithm to cluster nodes with processing score and GPS information. Consequently, since the server can determine a reasonable latency for a cluster, the CSA technique improves the overall throughput of federated learning while reducing false-positive dropouts.
- We proposed a novel additive sharing-based masking scheme robust to dropout nodes without using a (t, n) threshold secret sharing scheme. In particular, it allows nodes to verify the integrity and correctness of the masks received from other nodes for reliable aggregation. It also keeps local parameters private if each cluster has at least three nodes that are honest (no collusion with the server) nodes.

We briefly review the related work in Section 2 and explain a concrete description of the proposed model in Section 3. The security of the suggested model is analyzed in Section 4, and the simulated performance is analyzed in Section 5. Finally, we conclude the paper in Section 6.

2. Related Work

The main goal of secure aggregation is to protect the privacy of each user's private data from the server's aggregation under the context of a central server and multiple mobile nodes. A classic approach for secure aggregation is to use secure multiparty computation (MPC) protocols such as Yao's garbled circuits [5], secret sharing [6], and homomorphic encryption [7–10]. Protocols based on garbled circuits are suitable for a small number of parties, whereas protocols based on secret sharing or homomorphic encryptions can be applied to hundreds of users [14]. Secret sharing has been actively used in recently proposed secure aggregation protocols since it can make the aggregation robust to dropout users. However, secret-sharing-based protocols have the drawback of high communication costs because each node creates n shares and shares them with other nodes. Homomorphic encryption allows the aggregation to be performed on encrypted data in a public key infrastructure. However, the main weakness of homomorphic encryption is that it is computationally expensive and needs an additional trusted party. Protocols [7,8] based on threshold additively homomorphic cryptosystems can handle dropout users but require additional trust assumptions. Pairing-based schemes [10] also require a trusted dealer to set up the keys. The protocol outlined in [9] computes the sum by sequentially performing one round of interactions between the server and each client, but it does not deal with dropout users.

In order to prevent information leakage divulged by analyzing the differences between uploaded parameters from clients, the concept of differential privacy (DP) has been proposed [11,12]. K. Wei et al. [12] added artificial noises to parameters at the client side before aggregating. However, there is a tradeoff between the convergence performance and privacy protection levels, that is, a better convergence performance leads to a lower protection level. Another approach, which is very related to our solution, is to use pairwise

masks to hide the local parameters from the server. The pairwise blinding approach has been previously proposed in the studies in [15–18] and suggests different ways for dealing with client failure. K. Bonawitz et al. [13,14] proposed notable additive masking-based secure aggregation for federated learning. Users hide their local updates using paired perturbations, which will be canceled in the aggregation, so the server only obtains the correct sum of all the local updates. In an improved study [14], they solved the tolerance problem to dropout users using Shamir's secret sharing scheme. Our proposed secret sharing is also based on pairwise masking by additive secret sharing, but the way of dealing dropout users is different. We did not exploit Sharmir's secret sharing. In addition, our scheme clusters mobile nodes by processing time and performs cluster-based aggregation for computation and communication efficiency.

In order to reduce the overheads of secure aggregation, J. So et al. [19] proposed turbo aggregation that performs a circular aggregation for multiple groups. The group-based aggregation can reduce communication overheads because the sharing of masks and other data for secure aggregation is restricted to group members rather than to all the other nodes. It is similar to our scheme in that the aggregation is carried out on a group basis, but the turbo aggregation randomly partitions groups and operates the group aggregation in a circular way. On the other hand, the cluster in the proposed model is determined according to the process time and location of each node, and the cluster aggregation is independently performed in parallel. A. R. Elkordy and A. S. Avestimehr [20] suggested another group-based aggregation method for federated learning that allows the use of heterogeneous quantization according to communication resources in order to improve the communication performance. Networks are partitioned into groups, and user updates are also partitioned into segments. In addition, different levels of quantization are applied on segments during the aggregation. C. Hu et al. [21] proposed a cluster-based distributed aggregation model to resolve the bottleneck of centralized aggregation. The edge devices are grouped into clusters, and the aggregation is performed by some edge devices selected as cluster heads. Then, they are finally aggregated by the server. The authors proposed algorithms to find clusters that can minimize the maximum aggregation overhead. This is similar to our method in terms of the cluster-based aggregation. However, in our model, the server is trusted, so the aggregation is performed only on the server to assure the reliability of the aggregation. We believe that additional verifiable aggregation techniques are required to enable aggregation by edge devices. So, our model performs cluster-based aggregation to reduce the overhead, but the aggregation is performed only on the server. Another approach to reduce the overhead is to use a gradient sparsification method that sends only k parameters from the local gradient to the server. S. Lu et al. [22] proposed a top-k sparsification method for secure aggregation, which sends masked top-k parameters without exposing the coordinate information of the top-k parameters in a user model. However, there is a tradeoff between the accuracy of the training and communication efficiency.

Studies dealing with Byzantine users [23–25] in the learning model and studies on verifiable aggregation model [26,27] have also been proposed. L. He et al. [23] proposed a Byzantine robust secure gradient descent algorithm for a two-server model. It needs two honest and non-colluded servers, each of which carries out a secure two-party interactive protocol with mobile nodes. Z. Zhang et al. [25] proposed a lightweight Byzantine robust model with two servers. Each local result of the nodes is uploaded to two servers using a secret sharing method, so it achieves both local result protection and secure Byzantine robustness. On the other hand, J. So et al. [24] proposed a distance-based outlier detection approach for single-server federated learning, which can calculate the pairwise distance between local updates by sharing the masked updates and pairwise distances between the masked shares. This approach actually detects outliers and can work well if every node generates similar local updates. We do not think that outliers are necessarily Byzantine users. It is very challenging to detect Byzantine users who manipulate their updates in masked updates aggregation. We did not consider Byzantine users in this paper and left it for our future work. Regarding verifiable aggregation, Z. Yang et al. [26] provided a verification method to validate the weighted average aggregation result using a homo-

morphic hash function under the assumption that the federated learning server cannot be fully trusted. C. Hahn et al. [27] suggested a way to verify the correctness of local updates under a cross-device federated learning model. Our model basically assumes that all participants are honest-but-curious, so they do not manipulate the outputs. Instead, our model provides a verification method to validate the correctness and integrity of mask values shared between users. As mentioned above, verifying the correctness and validity of user local updates was not covered in this paper and will be performed in future work.

3. A Cluster-Based Secure Aggregation Model

Now, we describe the proposed cluster-based security aggregation (CSA) model in detail. First, the federated learning structure, system environment, and threat model assumed in this paper are briefly explained. In addition, we present the security requirements and define main functions constituting the CSA protocol, and then the detailed protocols for node clustering and secure aggregation algorithms are provided.

3.1. Background and Configuration

In this section, we briefly present the federated learning architecture used in this paper and describe our assumptions, system configuration, and notations. The federated learning system consists of a single central federated learning server and N mobile users (or nodes). In the rest of the paper, FS and U denote the federated learning server and a set of nodes, respectively, and each node is denoted by u_i. The local dataset (training data) of u_i has the property of non-IID and unbalanced distribution.

(1) Federated learning

The FS trains a global model $w \in \mathbb{R}^d$ with the dimension d using the data stored in mobile devices. This training process is used to minimize a global objective function $F(w)$:

$$\underset{w}{\operatorname{argmin}} F(w) \tag{1}$$

where $F(w) = \sum_{i=1}^{N} \frac{n_i}{n} F_i(w)$.

Here, N is the total number of mobile nodes, F_i is the local objective function of u_i, n_i is the private data size of u_i, and $n = \sum_i n_i$. The local objective function $F_i(w)$ of u_i for the global model w is defined as:

$$F_i(w) = \frac{1}{n_i} \sum_{j=1}^{n_i} f_i(w) \tag{2}$$

where $f_i(w) = l(x_i,\ y_i;\ w)$.

$f_i(w)$ is the loss of the prediction on example $(x_i,\ y_i)$ made with model parameters w [1].

For a fixed learning rate η, the FS trains the global model by iteratively performing the distributed stochastic gradient descent (SGD) method with currently available mobile nodes. At iteration t, the server shares the current global algorithm state (e.g., the current model parameters), w^t, with the mobile nodes. Each u_i then computes $\nabla F_i(w^t)$, which is the average gradient on its local data at the current model w^t and generates its local update w_i^{t+1}:

$$w_i^{t+1} := w^t - \eta \nabla F_i(w^t). \tag{3}$$

u_i iterates the local update multiple times before sending the update to the FS. Then, the FS aggregates these gradients and updates the global model for the next iteration:

$$\begin{aligned} w^{t+1} &:= \sum_{i=1}^{N} \frac{n_i}{n} w_i^{t+1} \\ &= w^t - \eta \sum_{i=1}^{N} \frac{n_i}{n} \nabla F_i(w^t) = w^t - \eta \nabla F(w^t) \end{aligned} \tag{4}$$

Since the loss gradient $\nabla F(w^t)$ can be rewritten as a weighted average across nodes, $\nabla F(w^t) = \sum_{i=1}^{N} \frac{n_i}{n} \nabla F_i(w^t)$.

(2) Communication

We basically assumed that all communication goes through the *FS*. Thus, all messages between nodes are first forwarded to the *FS*, and the *FS* sends them back to the corresponding nodes. However, this can be directly extended to node-to-node communication depending on the communication environment.

(3) System parameters

The *FS* generates a big prime number p, a public-key cryptosystem, and the server's private and public key pair, which are denoted as $<K_S{}^-, K_S{}^+>$. $PKE(K^+, M)$ represents a public key encryption for a message M with a public key K^+, and $PKD(K^-, C)$ denotes a public key decryption algorithm for a ciphertext C with a private key K^-.

(4) Registration

Before participating in federated learning, each node registers with the *FS*. The node creates an ID and a pair of private and public keys, and then registers it with the *FS*. The *FS* finally creates a shared symmetric key between the node and the *FS*, which is used for efficient data encryption between them. The shared key can be generated by the Diffie–Hellman key exchange protocol or randomly generated by the *FS*. In our model, the *FS* generates a random key, encrypts it with the node's public key, and sends it to the node. The *FS* securely manages IDs, public keys, and shared keys.

Table 1 summarizes the notations used in the rest of paper.

Table 1. Notations.

Notation	Description
u_i, U	The i-th node, where U is a set of all nodes
N	The total number of mobile nodes
n_i, n	The private data size of u_i, and $n = \sum_i n_i$
PS_i	The processing score of u_i
w^t	The global model parameter at iteration t
w_i^{t+1}	The local update of the u_i for w^t
$G_{i,j}$, G	The grid of row i and column j, where G is a set of all grids.
$U_{i,j}$	A set of nodes mapped to the grid $G_{i,j}$
C_i, C	The i-th cluster, where C is a set of all clusters.
r_i, R_i	A random integer that the FL server assigns to C_i and $R_i = g^{r_i} \bmod p$
$m_{j,k}$, $M_{j,k}$	A random mask that u_j generates for u_k where $M_{j,k} = g^{m_{j,k}} \bmod p$
S_i	The share of u_i for secure aggregation
IS_i	The intermediate sum of C_i

3.2. Problem Definition

The federated learning model protects the privacy of training data by letting the private training data remain on the user's mobile device. Instead, it requires only the local model parameters trained by each mobile device. However, it has been recently demonstrated that a server can still reconstruct the private data from local models using a model inversion attack [2–4]. Thus, secure aggregation, which does not expose the local model parameters during the server's aggregation for the federated learning, is necessary. The core of secure aggregation is for the federated learning server to know only the aggregated sum $S = \sum_{i=1}^{N} w_i$ of the local model parameters w_i without knowing any information about w_i. Another major issue to consider in secure aggregation is dropout users. Users who initially participate in federated learning may be dropped out during the federated learning due to network or device issues. Therefore, secure aggregation must tolerate dropout users, that is, the aggregation should be able to correctly reconstruct the

aggregated sum of the currently available users, even if dropout users occur. In addition, the decision on dropout can be erroneous, and a dropout user's response is not actually dropped out, but delayed, so it may be delivered to the server after the aggregation. Even in this case, no one should know any information about the local model of the dropout user from the delayed response. As mentioned earlier, it is not easy to distinguish between message drop and delay during each round of aggregation. Federated learning requires iterative aggregations, and the server cannot wait indefinitely for all responses from every single aggregation. Therefore, the server must be able to determine a reasonable latency to terminate a single aggregation, and this can be also used to distinguish between message drop and delay. The proposed CSA solves this problem simply and practically. After all, the communication latency to the server is mainly affected by the processing time of each device and the network conditions. The processing time of each device is determined by the size of the local dataset and the computational power required to train the dataset. On the other hand, it is not easy to measure the actual communication conditions between each mobile node and the server. Instead, we exploited two assumptions, i.e., (1) the processing delay of each device dominates over the communication delay and (2) the communication delay is proportional to the distance to the server. Accordingly, rather than measuring the actual network state, the geographic location information of each device is simply used to estimate the distance to the server. Based on this, the key strategy of the CSA technique first clusters mobile devices according to their processing time and geographic location and then performs the aggregation on a cluster basis. Since the processing time and communication distance of each node in a cluster can be estimated, the server can determine the appropriate latency for each cluster. In addition, the cluster-based strategy reduces the computational and communicational overheads for the secure aggregation because each node needs additional computation and communication to share random masks only for the same cluster members.

We considered an honest-but-curious threat model. All participants, including the *FS* and mobile nodes, act honestly in the communications. They do not deviate from the defined protocol but attempt to learn all the possible information from legitimately received messages. Under this threat model, the proposed aggregation is satisfied with the following security requirements:

- **Privacy of local datasets and updates:** All the data that each node holds in its local device and all the local learning parameters that are shared over the network must be confidential not only to the other nodes but also to the *FS*. The *FS* only knows the aggregated sum of all the local updates provided by all nodes. In addition, even if a particular user's update would be delivered to the *FS* after the aggregation, the *FS* and the other users cannot reconstruct the corresponding user's local parameters with the delayed data.
- **Tolerance to dropouts:** User updates can be dropped out during the communication due to network conditions and device issues. The *FS* should be able to compute a correct aggregated sum of the current active users even if dropout users occur.
- **Integrity of random masks:** Users create random masks and share them with other users to hide their actual local model parameters. In addition, these masks must be correctly removed during the local update aggregation. Therefore, users should be able to validate the correctness and integrity of given masks. In other words, users can be sure that the masks are created to be necessarily removed during the aggregation and that the masks are not modified during the communication.

We now describe the proposed CSA protocol and its main functions. The CSA protocol consists of two main parts: node clustering and secure aggregation. The node clustering is performed once at the beginning of the federated learning. It simply clusters nodes according to their processing scores and GPS information. After the clustering, the *FS* repeats the secure aggregation with nodes. The secure aggregation consists of the following key functions:

(1) Quantization: The CSA exploits an additive secret sharing defined over a finite field for a prime p. Thus, all operations in the CSA are carried out over $Z_p{}^*$. Since the local parameters are real numbers, the nodes need to quantize the real values to integers. To achieve this, we exploited So et. al.'s stochastic quantization strategy [24].
(2) Masking with additive secret sharing: The *FS* selects a random nonce for each cluster. Then, each node generates random masks for the other nodes in the same cluster and shares the encrypted masks with them. The random masks are created by an additive secret sharing method based on the cluster random nonce. Then, the nodes create their updates masked with those shares.
(3) Aggregation: The updates of nodes are first aggregated on a cluster basis. When dropout users occur in a cluster, the currently available nodes in the cluster perform a recovery phase. They modify the cluster sum by removing the masks of dropout users from the aggregated sum. After the recovery phase, the cluster sums are finally aggregated.

The detailed protocol for each function is described in the following sections.

3.3. Node Clustering

In this section, we describe the node clustering in detail. To achieve this, we first defined a processing score representing the total processing time of each node. Based on the size of dataset, the computation capacity *CC* and the processing score *PS* are defined as per Equation (5):

$$\begin{aligned} CC &= \text{floating point operations/second}, \\ PS &= \text{training data size (MB)}/CC \end{aligned} \tag{5}$$

Whenever a new federated learning starts, every node calculates *PS* and sends the *PS*, training data size, and GPS information to the *FS*. The key strategies for node clustering are:

- Each cluster must contain at least four nodes (the minimum cluster size denoted as δ is four).
- *PS* is divided into K levels (K is systemically predefined).
- The entire area where all nodes are distributed is divided into an $a \times b$ grid (a and b are systemically predefined), and the nodes are mapped to the grid by GPS information.
- Node clustering starts with the nodes closest to the *FS* according to the PS level. This repeats sequentially for the next neighbor nodes around the *FS* until all nodes are clustered.
- Clusters are finally rebalanced so that each cluster satisfies δ.

Let u_i's processing score be PS_i. PS_i's PS level is simply determined by the following Equation (6):

$$PSL(PS_i) = \left\lfloor \frac{PS_i \cdot K}{\max_{u_j \in U}(PS_j) - \min_{u_j \in U}(PS_j)} \right\rfloor + 1, \tag{6}$$

where K is the number of PS levels.

With the PS level, *FS* generates a grid, G, for nodes. For the entire area containing all nodes, the *FS* partitions the area into $a \times b$ cells for the predefined parameters a and b. $G_{i,j}$ denotes a cell at the i-th row and j-th column. The *FS* and nodes are mapped to the grid cells based on their GPS information. $U_{i,j}$ denotes a set of nodes mapped to $G_{i,j}$. Figure 1 shows an example of node distribution over a 6×5 grid. The number in a circle indicates the PS level of each node.

The *FS* creates K clusters that are initially empty. C_i denotes the cluster of level i for $1 \leq i \leq K$. The *FS* proceeds the node clustering by expanding the range of the neighbor around the server. Suppose that the cell where the *FS* is located is $G_{rf,cf}$. The initial neighbor is $G_{rf,cf}$ itself. For all the nodes in the neighbor, each node is allocated to a cluster of the same level as its own PS level. That is, if the PS level of a node is l, then the node is assigned to C_l. After the first round of clustering is complete, the next neighbor is selected with all the cells surrounding $G_{rf,cf}$ with an index difference of 1. These are $G_{rf-1,\,cf-1}$, $G_{rf-1,\,cf}$, $G_{rf-1,\,cf+1}$, $G_{rf,\,cf-1}$, $G_{rf,\,cf+1}$, $G_{rf+1,\,cf-1}$, $G_{rf+1,\,cf}$, and $G_{rf+1,\,cf+1}$. In this way, the next neighbor is changed by increasing the index by 1, and the node clustering is repeated for

all nodes in the next neighbor. The blue dotted line in Figure 1 represents the neighbor in each round. As shown in the figure, u_1, u_2, and u_3 are clustered in the first round according to the PS level, and then u_4 is clustered in the second round. The rest of nodes are clustered in the same way.

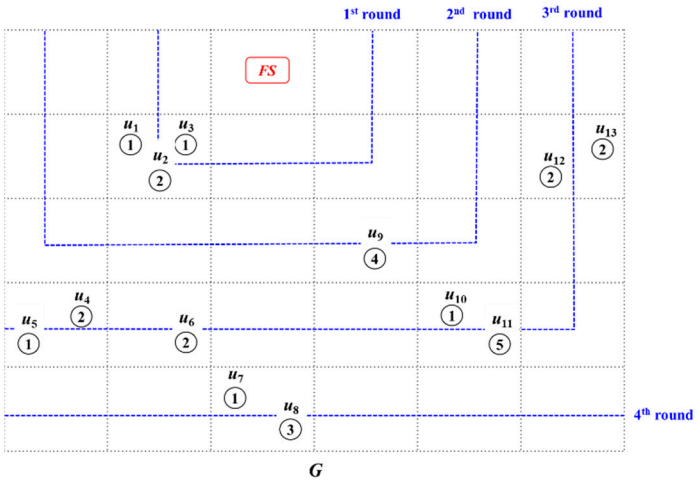

Figure 1. An example of node distribution over a grid.

Here, for each cluster, the nodes are hierarchically organized in the order in which they were clustered. Although all the nodes belonging to the same cluster have the same PS level, the distance to the *FS* is different, so hierarchical organization is required to classify the distance to the *FS* in the cluster. The detailed node clustering algorithm is shown in Algorithm 1 below.

Algorithm 1: Node Clustering

Input: G: a grid of nodes,
mr: the maximum row index,
mc: the maximum column index,
rf: the row index of the server cell,
cf: the column index of the server cell,
$U = \{u_1, \ldots, u_N\}$ and each u_i's GPS information and PS level.
Output: K clusters $C_1, \ldots, C_K$

```
create K empty clusters C_1, ..., C_K, where each C_i is hierarchically structured with an initial bucket B = ∅;
for each node u in U
    generate U_{i,j} using each u's GPS information and G;
for (D = 0; (rf + D <= mr || cf + D <= mc); D++)
    for each cluster C_i
            if (the leaf bucket of C_i ≠ ∅)
                create a new bucket B = ∅ and add B to C_i as a leaf bucket;
        r_0 = max(rf − D, 1);
        c_0 = max(cf − D, 1);
        r_1 = min(rf + D, mr);
        c_1 = min(cf + D, mc);
    for (r = r_0; r <= r_1; r++)
            for (c = c_0; c <= c_1; c++)
                if(|r − rf| == D || |c − cf| == D)
                    for each node u in U_{r,c}
                        l = u's PS level;
                        B_l = the leaf bucket of C_l;
                        B_l = B_l ∪ u;
return C_1, ..., C_K;
```

Figure 2 shows the results after clustering all nodes in Figure 1. Since the nodes of each cluster are hierarchically structured in the order that they are clustered, the nodes of C_1 have a three-tiered structure, and the nodes of C_2 have a two-tiered structure. On the other hand, C_3, C_4, and C_5 do not satisfy the minimum cluster size. Therefore, an additional cluster rebalancing process is required for all the clusters to meet the minimum cluster size requirement.

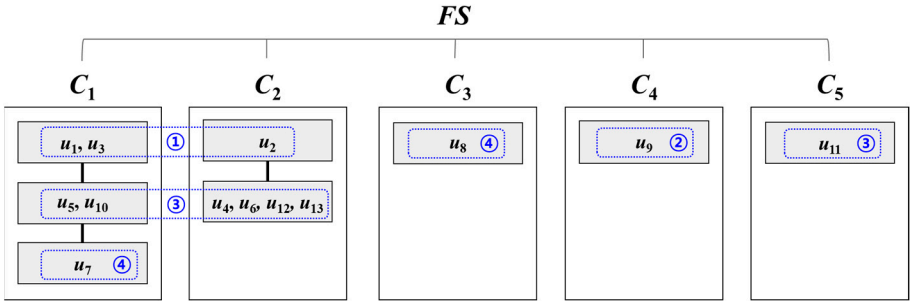

Figure 2. Results after clustering.

Once the clustering is complete, the *FS* forms the final clusters by merging the clusters whose size is smaller than δ. The merge begins from the cluster level K and repeats the following process. If C_K does not satisfy δ, and if C_{K-1} is sufficiently big, then some nodes of C_{K-1} are merged to C_K. Otherwise, C_K is merged to C_{K-1}. The detailed Algorithm 2 is given below.

Algorithm 2: Merge Clusters

```
while K > 1
    if |C_K| < δ
        l = δ − |C_K|;
        if (|C_{K−1}| − l) >= δ
            l nodes at the highest level in C_{K−1}'s node structure are merged to C_K;
        else
            C_{K−1} = C_{K−1} ∪ C_K;
            remove C_K;
    K = K − 1;
while |C_1| < δ
    C_n = the next-order cluster of C_1;
    C_1 = C_n ∪ C_1;
    remove C_n;
```

Figure 3 shows the final results after the cluster merging. First, $|C_5|$ is 1 and $|C_4|$ is 1, so u_{11} is merged into C_4. C_5 is discarded. Next, $|C_4|$ is still 2 and less than 4. $|C_3|$ is also 1, so u_9 and u_{11} in C_4 are merged into C_3. C_4 is discarded too. Now $|C_3|$ is 3, but it lacks one node to meet the requirements. $|C_2|$ is sufficiently big, although one node is moved to C_3, so u_{13} with the longest distance to the *FS* is merged into C_3. Since both C_2 and C_1 meet the requirements, the merge process is terminated. Consequently, C_1, C_2, and C_3 are finally created, and C_1 contains nodes <u_1, u_3, u_5, u_{10}, u_7>, C_2 contains nodes <u_2, u_4, u_6, u_{12}>, and C_3 contains nodes <u_2, u_4, u_6, u_{12}>.

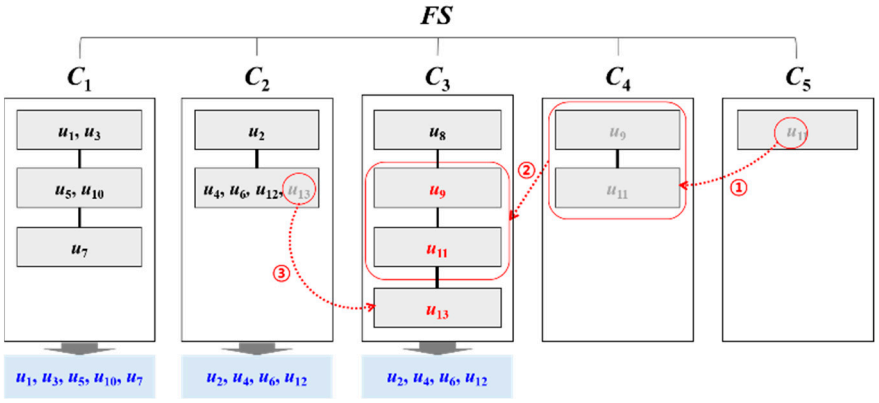

Figure 3. Final results after cluster merging.

After finishing the node clustering, the *FS* determines an appropriate latency (or latency level) for the dropout decision for each cluster. This can be set to three times the shortest response time in each cluster. The *FS* then sends the cluster ID and the list of the

nodes in the cluster to each node. The node list contains the node ID and the public key pair of each node belonging to the same cluster.

3.4. BCSA: A Basic Cluster-Based Secure Aggregation Model

Next, we describe the basic cluster-based secure aggregation protocol in detail. This protocol is denoted as *BCSA* in the rest of the paper. Aggregation of the local updates of the nodes is conducted on a cluster basis, and the *FS* eventually aggregates all the intermediate sums of the clusters. *BCSA* is defined by the following steps:

Step 1: Assignment of training weights and random nonce

Step 1 is performed only once before beginning the aggregation. The *FS* computes training weights for nodes and chooses random nonces for clusters. Let C_i be the i-th cluster and $|C_i|$ be the size of C_i, which is the number of nodes belonging to C_i. Since each node has a different training data size, the *FS* assigns training weights to the nodes according to the training data size. Let λ_j be the training weight of u_j in C_i. λ_j is determined as follows:

$$\lambda_j = \frac{n_j}{\sum_{u_k \in C_i} n_k}$$

where n_j is the data size of u_j.

The *FS* also chooses a random integer $r_i \bmod p$ for C_i, which is later used to generate random masks at each node. The *FS* delivers r_i along with λ_j to u_j in C_i as encrypted with the public key of u_j. The *FS* also publishes $R_i = g^{r_i} \bmod p$ to all nodes so that every node can verify the validity of r_i.

After Step 1 is finished, the following steps are repeated until the federated learning ends.

Step 2: Quantization

The local parameters, w_j, of each node are converted to integers by the stochastic rounding function proposed in [24]. For any integer $q \geq 1$, the stochastic rounding function is as follows:

$$Q_q(x) = \begin{cases} \frac{qx}{q}, & \text{with prob. } 1 - (qx - [qx]) \\ \frac{qx+1}{q}, & \text{with prob. } qx - [qx] \end{cases} \tag{7}$$

where $[x]$ is the largest integer less than or equal to x, and q is the number of quantization levels. For a mapping function $\phi : \mathbb{R} \rightarrow \mathbb{F}_p$, the quantized model is defined as follows:

$$W_j := \phi(q \cdot Q_q(\lambda_j \cdot w_j)), \tag{8}$$

$$\text{where } \phi(x) = \begin{cases} x, & \text{if } x \geq 0 \\ p + x, & \text{if } x < 0 \end{cases} \tag{9}$$

Step 3: Random masks generation and distribution

Each u_j in C_i generates random masks for all the other nodes in C_i. Let $m_{j,k}$ be a random mask for u_k where j and k are $1, \ldots, |C_i|$. u_j chooses $m_{j,k}$ satisfying the following Equation (10):

$$\sum_{\forall k \neq j} m_{j,k} \bmod p \;=\; r_i \bmod p \tag{10}$$

To achieve this, u_j firstly chooses $l - 1$ random positive integers $m_1, \ldots, m_{l-1}$ for modular p where $l = |C_i| - 1$ and then determines the final random integer m_l as $m_l = (r_i - \sum_{i=1}^{l-1} m_i)$. Here, m_l is not a value for modular p and can be negative. Thus, m_l is represented in the form of $x \cdot (p - 1) + r$ for an integer x and a positive residue r. Then, $\overline{m_l} := r \equiv m_l \bmod (p-1)$. u_j also generates a public mask $M_{j,k}$ for $m_{j,k}$'s validity verification as follows:

$$M_{j,k} := \begin{cases} g^{m_{j,k}} \bmod p & \text{if } m_{j,k} > 0, \\ g^{\overline{m_{j,k}}} \bmod p & \text{if } m_{j,k} < 0, \end{cases} \tag{11}$$

where $\overline{m_{j,k}} = m_{j,k} \bmod (p-1)$.

u_j encrypts $m_{j,k}$ with u_k's public key and publishes all the encrypted masks and public masks. The *FS* sends them back to nodes in the C_i. If $m_{k,j} < 0$, u_j sets $\overline{m_{k,j}} = m_{k,j} \bmod (p-1)$. Finally, u_j accepts $m_{k,j}$ if the following equation holds:

$$M_{k,j} = \begin{cases} g^{m_{k,j}} \bmod p & \text{if } m_{k,j} > 0, \\ g^{\overline{m_{k,j}}} \bmod p & \text{if } m_{k,j} < 0, \end{cases} \quad (12)$$
$$g^{r_i} = \prod_{\forall n \neq k} M_{k,n} \bmod p \text{ for } n = 1, \ldots, |C_i| - 1$$

If any mask is invalid or dropped during the communication, Step 3 is repeated until all the nodes share valid masks.

Step 4: Secure update generation

If all the masks are valid, u_j generates its secure update S_j for its local weight W_j as follows and sends S_j to the *FS*:

$$S_j = W_j + r_i - \sum_{\forall k \neq j} m_{k,j} \bmod p \quad (13)$$

Step 5: Cluster aggregation

For each cluster C_i, a different level of latency is determined to collect S_j. After collecting S_j for C_i, the *FS* determines a list of currently available users. If all nodes in C_i are available, the *FS* sends a message of "all available" to the nodes and computes the intermediate sum IS_i of C_i as follows:

$$\begin{aligned} TS_i &= \sum_{j=1}^{|C_i|} S_j \pmod p \\ &= \sum_{j=1}^{|C_i|} (W_j + r_i - \sum_{\forall k \neq j} m_{k,j}) \text{ for } 1 \leq k \leq |C_i| \\ &= \sum_{j=1}^{|C_i|} W_j + |C_i| \cdot r_i - \sum_{j=1}^{|C_i|} \sum_{\forall k \neq j} m_{k,j} \\ &= \sum_{j=1}^{|C_i|} W_j + |C_i| \cdot r_i - \sum_{k=1}^{|C_i|} \sum_{\forall j \neq k} m_{k,j} \text{ for } 1 \leq j \leq |C_i| \\ &= \sum_{j=1}^{|C_i|} W_j + |C_i| \cdot r_i - \sum_{k=1}^{|C_i|} r_i \\ &= \sum_{j=1}^{|C_i|} W_j + |C_i| \cdot r_i - |C_i| \cdot r_i \\ &= \sum_{j=1}^{|C_i|} W_j, \text{ and} \end{aligned}$$
$$IS_i = \phi^{-1}(TS_i),$$

where $\phi^{-1}(\cdot)$ is the dequantization function.

Eventually, IS_i is the average of the local weights w_j of all nodes.

Step 5-1: Recovery of aggregation: removing masks of dropout users

For each cluster C_i, if there are any dropout users, the *FS* determines a list of the current available users denoted as A_i and sends it back to all nodes. Each u_j in A_i replies with its signed confirmation messages to the *FS*, and the *FS* broadcasts all the signatures to all nodes. If all the signatures are valid, the currently available nodes carry out the recovery phase by removing the masks of the dropout users.

For example, suppose that $u_1 \sim u_4$ belong to a cluster C_1 and that u_2 and u_4 are dropout users, thus, S_2 and S_4 are dropped in the process of Step 4. The sum of S_1 and S_3 is defined as follows:

$$TS_1 = S_1 + S_3 = W_1 + W_3 + r_1 + r_1 - m_{2,1} - m_{3,1} - m_{4,1} - m_{1,3} - m_{2,3} - m_{4,3}$$

In order to recover the sum of w_1 and w_2, additional values of $m_{1,2}$, $m_{1,4}$, $m_{2,4}$, $m_{3,2}$, $m_{3,4}$, and $m_{4,2}$ are necessary. Here, $m_{1,2}$, $m_{1,4}$, $m_{3,2}$, and $m_{3,4}$ are created by u_1 and u_3, so those values can be easily obtained from u_1 and u_3. However, $m_{2,4}$ and $m_{4,2}$ are created by u_2 and u_4, respectively, and are delivered to u_2 and u_4, so u_1 and u_3 cannot know those values. This can be solved without revealing $m_{2,4}$ and $m_{4,2}$ directly. For u_2 and u_4, u_1 and

u_3 compute their reconstruction values RS_1 and RS_3 using the masks shared with u_2 and u_4 as follows:

$$RS_1 = m_{2,1} + m_{4,1} - m_{1,2} - m_{1,4},$$
$$RS_3 = m_{2,3} + m_{4,3} - m_{3,2} - m_{3,4},$$

Consequently, a new $TS_1{}'$ represents the sum of W_1 and W_3 correctly as follows:

$$TS_1' = TS_1 + RS_1 + RS_3$$
$$= W_1 + W_3 + r_1 + r_1 - m_{2,1} - m_{3,1} - m_{4,1} - m_{1,3} - m_{2,3} - m_{4,3} + m_{2,1} + m_{4,1}$$
$$- m_{1,2} - m_{1,4} + m_{2,3} + m_{4,3} - m_{3,2} - m_{3,4}$$
$$= W_1 + W_3 + r_1 + r_1 - m_{1,2} - m_{1,3} - m_{1,4} - m_{3,1} - m_{3,2} - m_{3,4}$$
$$= W_1 + W_3$$

We formalized the above. The nodes in C_i can be divided into two groups: active user group A_i and dropout user group D_i. If $D_i \neq \varnothing$, the nodes in A_i compute their reconstruction values RS_j using Equation (14) and send them to the *FS*:

$$RS_j = \sum_{u_k \in D_i} \left(m_{k,j} - m_{j,k} \right) \tag{14}$$

Then, the sum of all the updates of the final active users is computed by the following Equation (15):

$$TS_i = \sum_{u_j \in A_i} (S_j + RS_j) \mod p \tag{15}$$

Here, the training weight of each active user is a value calculated by considering the data of the dropout users, so it should be modified as a training weight for the data of only active users. Thus, the final sum of the active users is determined by the following Equation (16):

$$IS_i = \frac{\sum_{u_k \in C_i} n_k}{\sum_{u_j \in A_i} n_j} \cdot \phi^{-1}(TS_i) \tag{16}$$

Step 6: Final aggregation

Lastly, the *FS* obtains the S of all nodes by $S = \sum_{i=1}^{|C|} \frac{n\_A_i}{\sum_{j=1}^{|C|} n\_A_j} IS_i$ for all clusters, where $|C|$ is the number of clusters and $n\_A_i = \sum_{\forall u_j \in A_i} n_j$. S is the average of the local weights w_i of all active nodes.

3.5. FCSA: A Fully Secure Cluster-Based Aggregation Model

BCSA works correctly for dropout nodes. *BCSA*, however, allows the *FS* to know the actual local parameters of false dropout nodes when the nodes are determined to dropouts, but their updates are delivered to the *FS* after aggregation. For an example, suppose that u_d was determined as a dropout and u_d's update S_d is delivered to the *FS* after the other active users perform the recovery phase. In this case, the *FS* can know the actual local parameter W_d of u_d just by adding the reconstruction values of the active users to S_d of u_d. To solve this vulnerability, a fully secure cluster-based secure aggregation protocol was proposed. It is denoted as *FCSA* in the rest of the paper. *FCSA* uses another random secret chosen by a user to generate the secure update.

During the masking generation stage (Step 3 in Section 3.4), u_j in C_i chooses another random value α_j. At the secure update generation stage (Step 4 in Section 3.4), u_j generates its secure update S_j as follows:

$$S_j = W_j + r_i - \sum_{\forall k \neq j} m_{k,j} + \alpha_j \mod p \tag{17}$$

Unlike *BCSA*, the recovery phase is always necessary for all nodes to eliminate α_j, regardless of dropouts. If there are no dropouts, all nodes send their α_j to the *FS*. Otherwise,

each active user u_j in A_i sends its reconstruction values for the dropout nodes along with α_j. Even if u_d's S_d is delivered to the *FS* after the recovery phase, the actual local parameter W_d is still secure because it is hidden by u_d's random secret α_d.

Here, additional dropouts can occur even in the recovery phase. Let such a node be u_x, that is, the S_x of u_x is passed to the *FS* normally but the reconstruction value RS_x and secret α_x of u_x are dropped out in the recovery phase. If u_x turns out to be a dropout, the final active users send back the reconstruction values for all dropout users including u_x by Equation (14), and then the *FS* can compute the sum of the secure updates for the final active users by the following Equation (18):

$$TS_i = \sum_{\forall u_j \in A_i} (S_j + RS_j - \alpha_j) \bmod p \tag{18}$$

Even if a pair of RS_x and α_x of u_x is delivered to the *FS* late, u_x's local data W_x is still secure because S_x is still masked with the random values given from other nodes. We summarized the operation of *FCSA* with a simple example. For a cluster C_1 containing five nodes denoted as $u_1, \ldots, u_5$, suppose that u_2 and u_4 turned out to be a dropout in the initial cluster aggregation phase. Thus, S_1, S_3, and S_5 have been passed to *FS*. Then, u_1, u_3, and u_5 perform the recovery phase and should send their random secrets α_1, α_3, and α_5 and their reconstruction values RS_1, RS_3, and RS_5 generated by Equation (14) to the *FS*. RS_1, RS_3, and RS_5 are determined as follows:

$$RS_1 = m_{2,1} + m_{4,1} - m_{1,2} - m_{1,4}$$

$$RS_3 = m_{2,3} + m_{4,3} - m_{3,2} - m_{3,4}$$

$$RS_5 = m_{2,5} + m_{4,5} - m_{5,2} - m_{5,4}$$

Here, suppose that α_3 and R_3 are not delivered to the *FS* normally. Another dropout happens in the recovery phase. Then, the final active users u_1 an u_5 repeat the recovery phase by sending the modified reconstruction values ${R_1}^{(2)}$ and ${R_5}^{(2)}$ for all the dropout users u_2, u_4, and u_3 to the *FS*.

$${RS_1}^{(2)} = RS_1 + m_{3,1} - m_{1,3}$$

$${RS_5}^{(2)} = RS_5 + m_{3,5} - m_{5,3}$$

Finally, the *FS* obtains the sum of w_1 and w_5 of the final active users by calculating $S_1 + S_5 + {RS_1}^{(2)} + {RS_5}^{(2)} - \alpha_1 - \alpha_5$.

4. Security and Efficiency Analysis

In this section, we analyze the theoretical results of *FCSA* in terms of its robustness for dropout nodes, privacy of local parameters, and computational and communication efficiency of aggregation. This can be summarized as follows:

(1) Robustness to dropouts: *FCSA* is robust against dropout users. (There is no constraint for the number of active users.)
(2) Privacy of local parameters: *FCSA* guarantees the privacy of local parameters on each node if there are at least three honest active users in each cluster with a cluster size greater than or equal to four.
(3) Efficiency of secure aggregation: Let $\mathcal{C}$ be the average cluster size. The computation cost of a node is $O(\mathcal{C})$, and the *FS*'s cost is $O(N)$. The communication cost of node is $O(\mathcal{C})$, and the *FS*'s cost is $O(N\mathcal{C})$.

Next, we will first show that *FCSA* is robust against dropout users. To prove this, we will show that Equation (18) correctly derives the aggregated sum of the final active users.

For a cluster C and its random number r, let A be the group of final active users and D be the group of dropout users.

$$TS = \sum_{u_j \in A} (S_j + RS_j - \alpha_j)$$

$$\begin{aligned}
&= \sum_{u_j \in A} W_j + |A| \cdot r - \sum_{u_j \in A} \sum_{u_k \in U,\ k \neq j} m_{k,j} + \sum_{u_j \in A} \alpha_j + \sum_{u_j \in A} \sum_{u_d \in D} m_{d,j} - \sum_{u_j \in A} \sum_{u_d \in D} m_{j,d} - \sum_{u_j \in A} \alpha_j \\
&= \sum_{u_j \in A} W_j + |A| \cdot r - \sum_{u_j \in A} \left\{ \sum_{u_k \in U,\ k \neq j} m_{k,j} - \sum_{u_d \in D} m_{d,j} + \sum_{u_d \in D} m_{j,d} \right\} \\
&= \sum_{u_j \in A} W_j + \sum_{u_j \in A} \sum_{u_k \in U,\ k \neq j} m_{j,k} - \sum_{u_j \in A} \left\{ \sum_{u_a \in A,\ a \neq j} m_{a,j} + \sum_{u_d \in D} m_{d,j} - \sum_{u_d \in D} m_{d,j} + \sum_{u_d \in D} m_{j,d} \right\} \\
&= \sum_{u_j \in A} W_j + \sum_{u_j \in A} \left\{ \sum_{u_k \in U,\ k \neq j} m_{j,k} - \sum_{u_a \in A,\ a \neq j} m_{a,j} - \sum_{u_d \in D} m_{j,d} \right\} \\
&= \sum_{u_j \in A} W_j + \sum_{u_j \in A} \left\{ \sum_{u_a \in A,\ a \neq j} m_{j,a} + \sum_{u_d \in D} m_{j,d} - \sum_{u_a \in A,\ a \neq j} m_{a,j} - \sum_{u_d \in D} m_{j,d} \right\} \\
&= \sum_{u_j \in A} W_j + \sum_{u_j \in A} \sum_{u_a \in A,\ a \neq j} m_{j,a} - \sum_{u_j \in A} \sum_{u_a \in A,\ a \neq j} m_{a,j} \\
&= \sum_{u_j \in A} W_j
\end{aligned}$$

Next, we will show that *FCSA* guarantees privacy for each node's local parameters. We basically assumed honest-but-curious nodes and the *FS*. We assumed that they do not manipulate or forge their data but that some of them can collude with the *FS*. Any collusion node can provide its masking values to the *FS*.

Theorem 1. *For each cluster C that satisfies* $|C| \geq 4$*, if there are at least three honest (no collusion with FS) active users, FCSA guarantees the privacy of each node's local parameters.*

Proof. Let the non-collusive active user group be $H = \{h_1, h_2, h_3\}$ and the collusion user group be $B = U - H$. Let S_i be the secure update of n_i in A. S_i be defined as follows:

$$\begin{aligned}
S_i &= W_i + r - \sum_{u_j \in U,\ j \neq i} m_{j,i} + \alpha_i \\
&= W_i + \sum_{u_j \in U,\ j \neq i} m_{i,j} - \sum_{u_j \in U,\ j \neq i} m_{j,i} + \alpha_i \\
&= W_i + \sum_{u_h \in H,\ h \neq i} m_{i,h} + \sum_{u_c \in B \cap A,\ c \neq i} m_{i,c} + \sum_{u_k \in D} m_{i,k} - \sum_{u_h \in H,\ h \neq i} m_{h,i}
\end{aligned}$$

All the nodes in A also provide RS_i and α_i. Any node in B can provide all the masking values about the other nodes. Thus, the *FS* and the nodes in B can compute S_i' as follows:

$$\begin{aligned}
S_i' &= S_i + RS_i - \alpha_i - \sum_{u_c \in B \cap A,\ c \neq i} m_{i,c} + \sum_{u_c \in B \cap A,\ c \neq i} m_{c,i} \\
&= W_i + \sum_{u_h \in H,\ h \neq i} m_{i,h} - \sum_{u_h \in H,\ h \neq i} m_{h,i}
\end{aligned}$$

S_i' is still masked by the random secrets generated by honest nodes. $m_{i,h}$ and $m_{h,i}$ are securely shared between only n_i and n_h. Suppose that the *FS* has published S_i' to all the nodes in U. Here, we can consider two cases: (1) n_i belongs to B and n_h belongs to H, and (2) both n_i and n_h belong to H. For the first case, S_i' is determined as follows:

$$S_i' = W_i + m_{i,h_1} + m_{i,h_2} + m_{i,h_3} - m_{h_1,i} - m_{h_1,i} - m_{h_2,i} - m_{h_3,i}$$

All the nodes in B except n_i cannot know any masking value $m_{i,h}$ or $m_{h,i}$ because all the honest nodes do not open their masking values. Any honest node n_h in H can know $m_{i,h}$ or $m_{h,i}$ that n_h possesses but it cannot know the masking values of the other honest nodes. Therefore, it is impossible to reveal the actual w_i from all the opened information.

For the second case, let the three honest nodes be h, h_1, and h_2. The secure update of h is denoted as ${S_h}'$, which is defined as

$$S'_h = W_h + m_{h,h_1} + m_{h,h_2} - m_{h_1,h} - m_{h_2,h}.$$

All the nodes in B cannot know the masking values m_{h,h_j} that are shared between only the honest nodes. Thus, it is secure against all the nodes in B. For the other honest nodes h_1 and h_2, h_1 can know m_{h,h_1} and $m_{h_1,h}$ because h_1 possesses them, but it cannot know m_{h,h_2} and $m_{h_2,h}$. This is the same for h_2. Therefore, for both cases, W_i is always secure to all other nodes if there are at least three honest active nodes.

If the number of honest active nodes is less than three (e.g., two honest nodes marked h and h_1), ${S_h}'$ is defines as $S'_h = W_h + m_{h,h_1} - m_{h_1,h}$. When ${S_h}'$ is opened to h_1, h_1 can know the value of W_h because h_1 also possesses m_{h,h_1} and $m_{h_1,h}$. Therefore, at least three honest nodes are necessary to guarantee the privacy of the local parameters. □

Lastly, we analyzed the efficiency of *FCSA*. Table 2 summarizes the computation and communication costs of the nodes and the *FS* for each main operation of *FCSA*. Node clustering is performed only once by the *FS* at the beginning of the federated learning. After mapping the nodes to a grid according to the GPS, the *FS* assigns clusters sequentially from the nodes closest to the *FS*. The computational cost for the node clustering is O(N). During the aggregation, the main computational operations of each node side are mask generation, secure update generation, and reconstruction value generation. Since the aggregation is performed on a cluster basis, the computation costs for all these operations are proportional the size of the cluster (the number of nodes belonging the cluster). On the other hand, the main computational operations of the *FS* are training weight computation and the aggregation of local updates. In addition, all these operations are required for every node. Thus, the computational cost of the *FS* is proportional to the number of nodes. Therefore, the total computation costs of the nodes and the *FS* are O($\mathcal{C}$) and O(N), respectively, where $\mathcal{C}$ represents the average cluster size.

Table 2. Computational costs and communicational overheads of *FCSA*.

Operation		Computational Costs		Communicational Overheads	
		Node	**Server**	**Node to *FS***	***FS* to Nodes**
Node clustering		O(1)	O(N)	O(1)	-
Setup for aggregation—random nonce & training weight distribution to nodes		-	O(N)	-	O(N)
Local update generation	Share of masks	O($\mathcal{C}$)	-	O($\mathcal{C}$)	O($N\mathcal{C}$)
	Secure update generation	O($\mathcal{C}$)	-	O(1)	-
Aggregation	All active nodes (no dropouts)	-	O(N)	O(1)	-
	Recovery phase	O($\mathcal{C}$)	O(N)	O(1)	-
Total cost for aggregation		O($\mathcal{C}$)	O(N)	O($\mathcal{C}$)	O($N\mathcal{C}$)

$\mathcal{C}$: average cluster size; N: total number of nodes.

Next, in our model, all communications go through the *FS*. The most communicationally expensive operation of the node side is to send masks to the *FS*. For a cluster C, each node in C sends $(|C| - 1)$ masks to the *FS*. Thus, the communication cost of a node is proportional to the size of cluster and is O($\mathcal{C}$). On the side of the *FS*, the most communicationally expensive operation is to distribute the masks to all nodes. The FS sends $(|C| - 1)$ masks to each node in each cluster C. Since the FS must repeat this operation for all clusters, the FS eventually sends $(\mathcal{C} - 1)$ masks to N nodes. Thus, the communication cost of the FS is O($N\mathcal{C}$).

5. Experimental Results

In this section, we analyzed the simulated performance of the proposed CSA. We particularly evaluated the accuracy and overall processing time of our model using the MNIST [28] database for various federated learning situations. We compared the accuracy of our cluster-based learning model with a single centralized learning model according to quantization levels, training weights, and different dropout situations. In addition, we analyzed the overall processing time under various dropout situations.

5.1. Simulation Setup

We used the MNIST database for our experiment. The MNIST database contains 28×28 grayscale images of 10 digits and consists of a training set of 60,000 images along with a test set of 10,000 images. For an individual training run, a two-layer CNN model with 5×5 convolution layers (the first with sixteen channels, the second with thirty-two channels, each followed with 2×2 max pooling), ReLU activation, and a final softmax output layer was used. We conducted our experiments using Python and implemented the learning architecture using PyTorch framework.

The MNIST data were distributed into 100 nodes in a non-IID way. First, the nodes were evenly divided into four clusters (twenty-five nodes per cluster) with different PS levels. The PS level, denoted as *PSL*, had a value from 1 to 4. *PSL* 1 represents the group with the shortest response time, and *PSL* 4 represents the group with the longest response time. Accordingly, the nodes in the *PSL* 1 cluster were allocated a small amount of training data, whereas the nodes in the *PSL* 4 cluster were assigned a relatively large amount of training data. So, we assigned 100 randomly chosen examples to each node with *PSL* 1, 400 examples to each node with *PSL* 2, 700 examples to each node with *PSL* 3, and 1000 examples to each node with *PSL* 4. In total, 55,000 data items were used for the training in the experiment.

The client nodes were implemented by creating 100 threads on a PC configured with RTX 3080 GPU, Intel(R) i7-12700K 3.61GHZ CPU, 32 GB memory, and Windows 11. The server PC was configured with an RTX 3080 GPU, Intel(R) i9-11900F 2.50GHZ CPU, 1TB SSD, 64 GB memory, and Windows 11. Since the nodes were created as threads on the same PC, the actual communication latency and computing power of the nodes were the same. Only the actual local training times of the nodes were different due to the different sizes of the training data. Therefore, the single round latency per cluster was arbitrarily set to 20 s for the PSL 1 cluster, 25 s for the PSL 2 cluster, 30 s for the PSL 3 cluster, and 35 s for the PSL 4 cluster. This represented the maximum waiting time when dropout occurs. If dropout did not occur, the next round was performed as soon as responses from all nodes were collected. For the federated learning, the FS basically performed 100 iterations of local weight updates with nodes. The main experimental parameters and values are summarized in Table 3 below.

Table 3. Experimental parameters and values.

Parameters	Values
The total number of nodes/ the number of nodes per each cluster	100/ 25
The number of clusters (*C*)	1, 4
Quantization level (*qLevel*)	30, 100, 300
Dropout rate (*dr*)	0%, 30%, 50%
The size of prime *p*	15 bits
PS level (*PSL*)	1 ~ 4
Training data ratio according to PS level	PSL_1: 4.5%, PSL_2: 18%, PSL_3: 32%, PSL_4: 45.5%
The number of iterations for federated learning	100

5.2. Simulated Performance

In the experiment, *FCSA* with four clusters (C = 4) was used as the cluster-based federated learning model, while *FCSA* with a single cluster (C = 1) was used as the single centralized learning model. Before analyzing the accuracy of *FCSA*, when we simply performed the federated learning on 100 nodes without applying the proposed CSA, the accuracy was about 90.5%. Since *FCSA* requires quantization for mask sharing and validation, we first analyzed the accuracy of *FCSA* according to the quantization level. Figure 4 shows the accuracy of the proposed *FCSA* according to different quantization levels (qLevel). The simulated results showed the best accuracy when qLevel was 300 for both situations, and *FCSA* with C = 4 showed better accuracy than *FCSA* with C = 1. When no dropout occurred, the accuracy of *FCSA* with C = 4 was 91.08%, while that of *FCSA* with C = 1 was 87%. On the other hand, when qLevel was 100, the accuracy of *FCSA* with C = 4 was 88.3%, but the accuracy of *FCSA* with C = 1 was about 40%. From the results, we can see that the quantization level greatly affected the learning accuracy. However, using an appropriate quantization level could prevent the loss of accuracy due to quantization, as shown by the results showing that the accuracy of *FCSA* with C = 4 at a qLevel of 300 was better than that of the simple federated learning without quantization. In the federated learning model, since the local data size of each device was small, relatively small quantization values were sufficient to output accurate learning results. However, the centralized learning model required large quantization values, as it worked with the full data collected from the local devices.

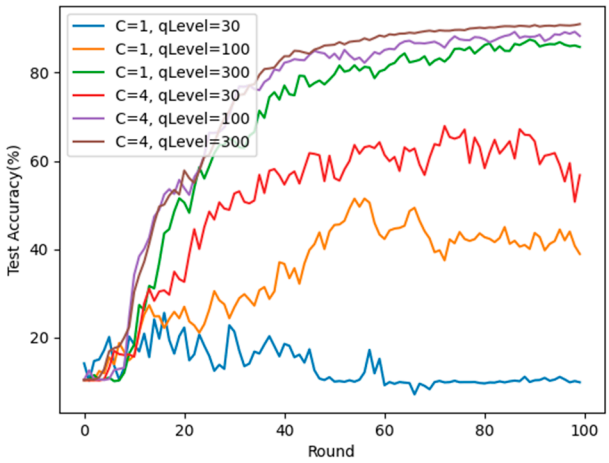

Figure 4. Accuracy according to the quantization levels.

Table 4 shows the accuracy according to the dropout rate (*dr*). In this experiment, for *FCSA* with C = 1, the nodes equal to the dropout rate among all the nodes were randomly removed from the federated learning. On the other side, in the case of *FCSA* with C = 4, the dropout nodes were selected per cluster. In other words, if *dr* = 30%, for each cluster, 30% of the nodes were randomly removed from the learning. It should be noted that the accuracy of *FCSA* with C = 1 decreased as the dropout rate increased, whereas the accuracy of *FCSA* with C = 4 was not significantly affected by the dropout rate. *FCSA* with C = 4 outputted similar accuracies regardless of the dropout rate. This result was meaningful. In both cases, the actual amount of training data used for the federated learning was the same, but the accuracy was different. In the case of *FCSA* with C = 4, the data size used for local learning was different for each cluster. However, even if *dr* = 50%, it was noted that the accuracy of the overall learning did not deteriorate because the remaining nodes still produced good learning results in the C_3 and C_4 groups that performed local learning with relatively large amounts of data.

Table 4. Accuracy according to different dropout rates.

Dropout Rate (*dr*)	*FCSA* with $C = 4$		*FCSA* with $C = 1$	
	50 Round Accuracy (%)	100 Round Accuracy (%)	50 Round Accuracy (%)	100 Round Accuracy (%)
$dr = 0\%$	86.22	91.08	80.13	87
$dr = 30\%$	86.35	91.54	75.7	86.12
$dr = 50\%$	85.02	91.25	72.08	79.7

Figure 5 shows the difference in the accuracy between applying and not applying the training weights (λ) to *FCSA* with $C = 4$. When the training weights were not applied to *FCSA* with $C = 4$, the accuracy dropped slightly to 90.01%. Since *FCSA* with $C = 4$ was not affected by the dropout rate per cluster, the accuracy was compared when cluster-based dropout occurred. Figure 6 shows the results. When all the nodes belonging to C_1 were dropped out, the accuracy was around 91.8%. From this, it can be seen that the local learning result of the nodes belonging to the C_1 cluster had little effect on the overall learning, and the overall learning accuracy was rather increased by the nodes belonging to the C_3 and C_4 clusters based on sufficient data. On the other hand, when C_4 was dropped out, the accuracy dropped to 89.4%. When the training weights were not applied and C_4 was dropped out, the accuracy decreased to 87.11%. From the experiment results, we can conclude that an accurate local model based on sufficient data was the most important factor for increasing the accuracy of the overall federated learning.

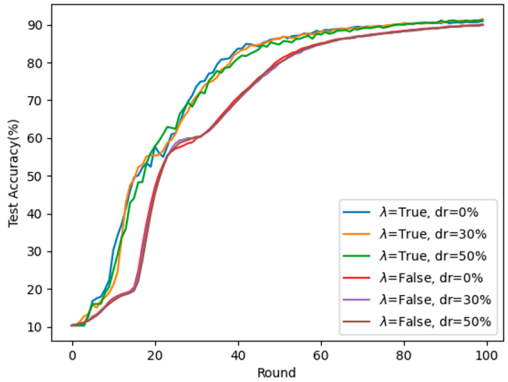

Figure 5. Accuracy by dropout rate and training weights (λ).

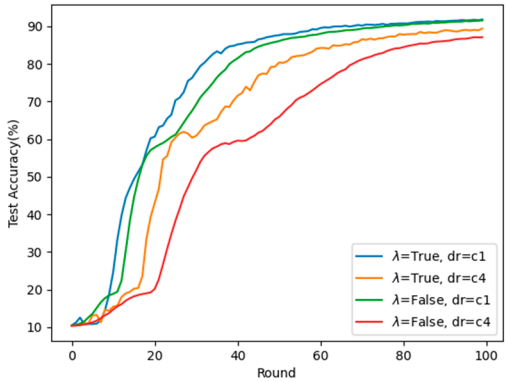

Figure 6. Accuracy by cluster-based dropout and training weights (λ).

Lastly, we analyzed the total running time according to different dropout situations. Figure 7 shows the overall running time (100 rounds run time) when cluster-based dropout occurred. When C_1 was dropped out, the overall time was about 1796 s, and when C_4 was dropped out, it was about 4588 s. This was because the latency was different depending on

the cluster level, and the total execution time increased proportionally accordingly. Figure 8 compares the total execution time of *FCSA* with $C = 1$ and *FCSA* with $C = 4$. When dropout did not occur, the running time of *FCSA* with $C = 1$ was about 1878 s, and the running time of *FCSA* with $C = 4$ was about 1815 s. The difference was negligible. When dropout occurred, the running time of *FCSA* with $C = 1$ was about 4872 s, while the running time of *FCSA* with $C = 4$ was 4011 s. *FCSA* with C = 1 applied the longest latency to all dropout nodes, whereas *FCSA* with C = 4 applied a different latency to dropout nodes depending on the cluster, so the total execution time of *FCSA* with C = 4 was relatively reduced.

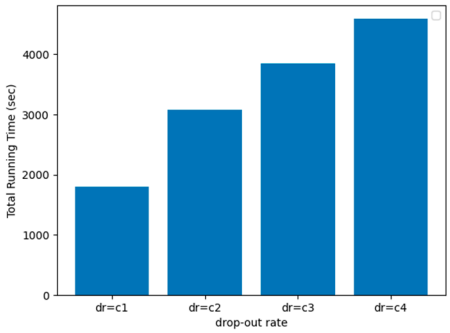

Figure 7. Total running time by cluster-based dropout.

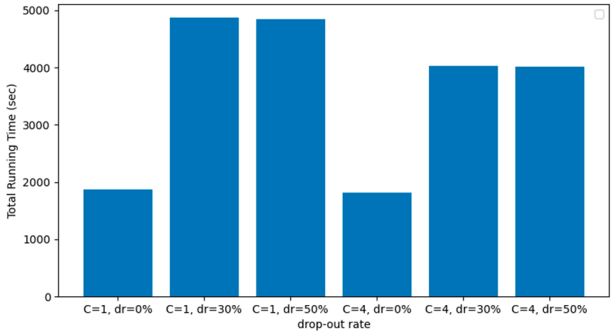

Figure 8. Total running time by dropout rate per cluster.

6. Conclusions and Future Work

The purpose of our study was to propose a practical secure aggregation model that protects the privacy of local updates, is robust against dropouts, reduces computation and communication costs, and reduces the overall running time in federated learning using heterogeneous devices. To achieve this, we proposed a new cluster-based secure aggregation (CSA) strategy that effectively handles dropouts and reduces overhead in situations where the size of the training data, the computing power, and the communication distance of each node are different.

To deal with dropout nodes, the server should first be able to detect dropout occurrence. In other words, the server must determine within a reasonable amount of time whether a non-response of some nodes is due to communication delay or message-drop. Increasing the waiting time to receive responses from nodes can slow down the entire federated learning time as it has to interact with nodes repeatedly.

The proposed CSA method clusters nodes with similar response times and performs cluster-by-cluster aggregation of local updates on nodes. To achieve this, we suggested a grid-based clustering algorithm that clusters nodes according to their processing levels and locations. In the model, the server can estimate an appropriate latency based on the response time of nodes per cluster and can treat unresponsive nodes as dropout nodes after the latency period has elapsed. Therefore, the proposed CSA method provides a reasonable solution that can reduce the overall federated learning time while increasing the decision accuracy for dropout.

We also proposed a new additive sharing-based masking technique to securely aggregate the local updates of nodes. It is robust to dropout users and protects the privacy of local model parameters if each cluster has at least three honest nodes (not colluded with the federated learning server). We theoretically proved the robustness and security of the proposed aggregation algorithm. Specifically, the proposed masking scheme allows nodes to publicly validate the correctness and integrity of masks created by others. The masks that are randomly created on nodes must be removed during the aggregation at the server for correct aggregation. Therefore, each node must be able to proactively check that the given masks can be eliminated later before generating a secure update with the masks. In the proposed masking scheme, nodes also publish public masks along with random masks securely delivered to other nodes, so the integrity of the given masks can be verified with the public masks based on a discrete logarithm problem. In addition, the masks of each node are created based on a random nonce (cluster nonce) assigned to each cluster. Thus, each node can easily and publicly validate the correctness of the masks created by others by testing that the multiplication of the public masks is equal to the public cluster nonce without knowing the actual masks delivered to others. The proposed masking technique effectively solves the mask verification problem using the discrete logarithm problem to ensure reliable aggregation. Here, the quantization of local parameters is required to use the discrete logarithm in the protocol, and the quantization greatly affects the learning accuracy. However, our simulated results showed that the loss of accuracy due to the quantization could be prevented by using an appropriate quantization level. In our experiments, there was no loss of accuracy when using a quantization level of 300 for *FCSA* with four clusters.

The proposed CSA method also reduces the computational and communication overhead by allowing nodes to share aggregation-related data only with nodes belonging to the same cluster. Supposing that $\mathcal{C}$ is the average cluster size and N is the number of nodes, each node's computational and communication costs are both O($\mathcal{C}$). On the other hand, the server's computational cost is O(N), while the communication cost is O($N\mathcal{C}$). We evaluated the performance of the proposed model using with the MNIST dataset. In the simulation for one hundred nodes, *FCSA* with four clusters outputted about a 91% learning accuracy regardless of the dropout rate, whereas a centralized *FCSA* (with one cluster) outputted about a 87% accuracy without dropout. For a dropout ratio of 50%, *FCSA* with one cluster dropped to about a 79% accuracy. In a centralized federated learning model, the accuracy is highly dependent on the dropout rate. However, in our model, a dropout rate of 50% hardly changed the accuracy. This was because the remaining nodes in the high-PS-level clusters still produced good learning results with relatively large amounts of data. The results showed that the proposed CSA method was suitable for federated learning using heterogeneous devices with different sizes of training data.

Our model basically assumes that all participants, including the federated learning server and nodes, are honest, so it does not separately verify each user's local updates and the server's aggregation result. However, the server and nodes cannot be fully trusted in the real world, so the proposed CSA method is vulnerable to Byzantine users. The CSA method also has a drawback in that a bottleneck can occur on the server. Therefore, we will conduct further research on verifiable cluster-based aggregation that can validate user results and aggregated results. With the verifiable aggregation, the proposed CSA method can be extended to a cross-device federated learning model, in which reliable and powerful nodes act like cluster heads that perform intermediate aggregation on behalf of the server. So, the bottleneck of the server can be also resolved.

We ultimately aim to develop a federated learning platform for medical diagnosis. Since medical data is the most-sensitive personal data, hospitals are extremely reluctant to expose it to the outside, so federated learning through secure aggregation is suitable for developing a medical diagnosis model based on medical data scattered across hospitals. In particular, each hospital deals with different diseases, and the size of the hospital and the corresponding patient data are different, and the diagnosis results are also different. Therefore, it is necessary to cluster hospitals according to the type of disease, the size of the hospital, the size of patient

data, and the level of care, and to perform cluster-based federated learning. In the future, we plan to apply the proposed model to actual medical data and propose an improved cluster-based secure aggregation model suitable for medical data.

Author Contributions: Conceptualization, J.K., G.P., M.K. and S.P.; methodology, J.K., G.P., M.K. and S.P.; software, J.K., G.P. and M.K.; validation, J.K., G.P. and S.P.; formal analysis, S.P.; writing—original draft preparation, S.P.; writing—review and editing, S.P.; supervision, S.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2021R1F1A1063172).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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

References

1. McMahan, H.B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, FL, USA, 9–11 May 2017; Volume 54. [CrossRef]
2. Zhu, L.; Liu, Z.; Han, S. Deep leakage from gradients. *arXiv* **2019**, 14774–14784. [CrossRef]
3. Wang, Z.; Song, M.; Zhang, Z.; Song, Y.; Wang, Q.; Qi, H. Beyond inferring class representatives: User-level privacy leakage from federated learning. In Proceedings of the IEEE INFOCOM, Paris, France, 29 April–2 May 2019; pp. 2512–2520. [CrossRef]
4. Geiping, J.; Bauermeister, H.; Dröge, H.; Moeller, M. Inverting gradients—How easy is it to break privacy in federated learning? In Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Online, 6–12 December 2020. [CrossRef]
5. Yao, A.C. Protocols for secure computations. In Proceedings of the 23rd IEEE Annual Symposium on Foundations of Computer Sciecne (sfcs 1982), Chicago, IL, USA, 3–5 November 1982; pp. 160–164. [CrossRef]
6. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613. [CrossRef]
7. Leontiadis, I.; Elkhiyaoui, K.; Molva, R. Private and dynamic timeseries data aggregation with trust relaxation. In Proceedings of the International Conferences on Cryptology and Network Security (CANS 2014), Seoul, Korea, 1–3 December 2010; Springer: Berlin/Heidelberg, Germany, 2014; pp. 305–320. [CrossRef]
8. Rastogi, V.; Nath, S. Differentially private aggregation of distributed time-series with transformation and encryption. In Proceedings of the ACM SIGMOD International Conference on Management of data (SIGMOD 10), Indianapolis, IN, USA, 6–10 June 2010; pp. 735–746. [CrossRef]
9. Halevi, S.; Lindell, Y.; Pinkas, B. Secure computation on the Web: Computing without simultaneous interaction. In *Advances in Cryptology—CRYPTO 2011*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 132–150. [CrossRef]
10. Leontiadis, I.; Elkhiyaoui, K.; Önen, M.; Molva, R. PUDA—Privacy and Unforgeability for Data Aggregation. In *Cryptology and Network Security. CANS 2015*; Springer: Cham, Switzerland, 2015; pp. 3–18. [CrossRef]
11. Geyer, R.C.; Klein, T.; Nabi, M. Differentially private federated learning: A client level perspective. In Proceedings of the NIPS 2017 Workshop: Machine Learning on the Phone and other Consumer Devices, Long Beach, CA, USA, 8 December 2017. [CrossRef]
12. 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]
13. Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedoney, A.; McMahan, H.; Patel, S.; Ramage, D.; Segal, A.; Seth, K. Practical Secure Aggregation for Federated Learning on User-Held Data. *arXiv* **2016**. [CrossRef]
14. 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 ACM SIGSAC Conferences on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1175–1191. [CrossRef]
15. Ács, G.; Castelluccia, C. I have a DREAM! (DiffeRentially privatE smArt Metering). In *Information Hiding. IH 2011*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 118–132. [CrossRef]
16. Goryczka, S.; Xiong, L. A comprehensive comparison of multiparty secure additions with differential privacy. *IEEE Trans. Dependable Secur. Comput.* **2017**, *14*, 463–477. [CrossRef]
17. Elahi, T.; Danezis, G.; Goldberg, I. Privex: Private collection of traffic statistics for anonymous communication networks. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; pp. 1068–1079. [CrossRef]
18. Jansen, R.; Johnson, A. Safely Measuring Tor. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 1553–1567. [CrossRef]

19. So, J.; Güler, B.; Avestimehr, A.S. Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning. *arXiv* **2020**, arXiv:2002.04156. [CrossRef]
20. Elkordy, A.R.; Avestimehr, A.S. HeteroSAg: Secure Aggregation with Heterogeneous Quantization in Federated Learning. *IEEE Trans. Commun.* **2022**, *70*, 3151126. [CrossRef]
21. Hu, C.; Liang, H.; Han, X.; Liu, B.; Cheng, D.; Wang, D. Spread: Decentralized Model Aggregation for Scalable Federated Learning. In Proceedings of the 51st International Conference on Parallel Processing (ICPP'22), Bordeaux, France, 29 August–1 September 2022; pp. 1–12. [CrossRef]
22. Lu, S.; Li, R.; Liu, W.; Guan, C.; Yang, X. Top-k sparsification with secure aggregation for privacy-preserving federated learning. *Comput. Secur.* **2023**, *124*, 102993. [CrossRef]
23. He, L.; Karimireddy, S.; Jaggi, M. Secure byzantine robust machine learning. *arXiv* **2020**, arXiv:2006.04747. [CrossRef]
24. So, J.; Güler, B.; Avestimehr, A.S. Byzantine-Resilient Secure Federated Learning. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 2168–2181. [CrossRef]
25. Zhang, Z.; Wu, L.; Ma, C.; Li, J.; Wang, J.; Wang, Q.; Yu, S. LSFL: A Lightweight and Secure Federated Learning Scheme for Edge Computing. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 365–379. [CrossRef]
26. Yang, Z.; Zhou, M.; Yu, H.; Sinnott, R.O.; Liu, H. Efficient and Secure Federated Learning With Verifiable Weighted Average Aggregation. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 205–222. [CrossRef]
27. Hahn, C.; Kim, H.; Kim, M.; Hur, J. VerSA: Verifiable Secure Aggregation for Cross-Device Federated Learning. *IEEE Trans. Dependable Secur. Comput.* **2023**, *20*, 36–52. [CrossRef]
28. LeCun, Y.; Cortes, C.; Burges, C.J. MNIST Handwritten Digit Database. 2010. Available online: http://yann.lecun.com/exdb/mnist (accessed on 6 January 2023).