ClusterGuard: Secure Clustered Aggregation for Federated Learning with Robustness

Yulin Zhao, Zhiguo Wan, *Senior Member, IEEE*, Zhangshuang Guan, Guannan Li and Miao Guo

***Abstract*—Federated Learning, as a multi-party machine learning paradigm, has garnered significant attention, but model updates may still leak sensitive information. Secure aggregation protocols are considered an effective solution for privacy protection in Federated Learning. However, in large-scale federated learning systems, designing efficient and practical secure aggregation remains a critical challenge. Moreover, while secure aggregation effectively conceals model updates, it unintentionally complicates the detection and mitigation of poisoning attacks, thereby exposing the system to vulnerabilities from both data and model poisoning.**

To address these challenges, we propose ClusterGuard, a secure clustered aggregation scheme. ClusterGuard leverages Verifiable Random Function (VRF) to ensure fair and transparent client clustering. Within each cluster, it employs a lightweight key-homomorphic masking mechanism combined with verifiable secret sharing to enable secure and efficient aggregation. Furthermore, we design a dual filtering mechanism based on cosine similarity and norm to effectively detect and resist poisoning attacks. We provide two variants of ClusterGuard for both client-server and decentralized environments with blockchains, respectively.

Extensive experiments on standard datasets demonstrate that ClusterGuard achieves over $2\times$ efficiency improvement compared to advanced secure aggregation methods. Even with 20% of clients being malicious, the trained model maintains accuracy comparable to the original model, outperforming state-of-the-art robustness solutions. ClusterGuard provides a more efficient, secure, and robust solution for practical federated learning.

***Index Terms*—Federated learning, Secure aggregation, Byzantine robustness**

I. Introduction

In recent years, amidst the rising importance of data privacy and security, federated learning (FL) has emerged as a solution for collaborative model training among multiple data custodians without sharing raw data. While FL prevents direct data exposure to third parties, it still carries significant privacy risks because the model updates (gradients) still leak sensitive user privacy, as highlighted in [1]. Thus, protecting model updates in FL training is paramount, underscoring the urgent need for a secure aggregation framework.

Currently, quite a few schemes have been proposed to address this pressing need. The pioneering work is due to Bonawitz et al. from Google [2], along with a series of subsequent works [3]–[6]. Their key idea is to establish pairwise opposite masks for each pair of clients to hide their model parameters, and these masks can be cancelled completely in aggregation. Assume a set of clients $\mathcal{U}$ with their local data intend to perform federated learning together, and each client $u \in \mathcal{U}$ performs local training to obtain his local model update $\mathbf{x}_u$, where the elements of $\mathbf{x}_u$ are in $\mathbb{Z}_R$ for some R. Then we can add the mask as follows to each $\mathbf{x}_u$:

$$\mathbf{y}_u = \mathbf{x}_u + \mathbf{PRG}(b_u) + \sum_{v\in\mathcal{U}:u<v} \mathbf{PRG}(s_{u,v}) - \sum_{v\in\mathcal{U}:u>v} \mathbf{PRG}(s_{v,u}) \mod R \quad (1)$$

where **PRG** is a pseudorandom generator, $s_{u,v}$ is the shared seed between clients u and v used to generate pairwise masks, and b_u is another random seed to generate individual masks. Individual masks effectively prevent potential privacy leakage in case of delayed message transmissions.

However, designing efficient and practical secure aggregation protocols remains a challenge. Each client needs to compute and share pairwise masks with other clients, which leads to increased complexity and resource demands. Bell et al. [3] and ACORN [7] adopted the Distributed Graph Generation approach to effectively reduce the communication overhead required by clients. But in the event of client dropouts, the server must reconstruct numerous pairwise masks, further hindering its practicality in large-scale federated learning (FL). Liu et al. [8], SASH [9], and LERNA [10] propose utilizing homomorphic pseudorandom generators to optimize the process into a *single* mask. However, their constructions face significant practical challenges: Liu et al.'s approach requires brute-forcing discrete logarithms, resulting in a substantial computational burden; SASH introduces additional communication rounds, with computation results containing unavoidable errors; and LERNA relies on a dedicated committee to process dropouts, which may be impractical in real-world scenarios.

More importantly, it is even more challenging to design countermeasures to poisoning attacks in secure aggregation protocols. Secure aggregation protocols, while concealing model updates, also increase the difficulty of detecting poisoning attacks. Rofl [11] and ACORN [7] leverage zero-

This work was supported by the National Natural Science Foundation of China under Grant (Nos. U22B2032, 62272425). (Corresponding author: Zhiguo Wan.)

Yulin Zhao is with Zhejiang Lab, Hangzhou, 311121, China, Institute of Software Chinese Academy of Sciences, Beijing 100190, China, and Hangzhou Institute for Advanced Study, University of Chinese Academy of Sciences, Hangzhou 310024, China (e-mail: zhaoyulin22@mails.ucas.ac.cn).

Zhiguo Wan is with Zhejiang Lab, Hangzhou, 311121, China (e-mail: wanzhiguo@zhejianglab.com).

Zhangshuang Guan is with the College of Computer Science and Technology, Zhejiang University, Hangzhou, 310027, China (e-mail: guanzs@zju.edu.cn).

Guannan Li is with School of Aerospace Engineering, Tsinghua University, Beijing, China (e-mail: 18311081686@163.com).

Miao Guo is with Network Investment (Beijing) Technology Co.Ltd., China (e-mail: gagamazel@163.com).

knowledge proof to ensure privacy and impose constraints on the norm of model updates. However, this approach introduces significant additional communication and computation overhead due to zero-knowledge proofs. Moreover, a single norm constraint can only defend against limited model poisoning attacks but not data poisoning attacks. Zhang et al. [12] proposed SAFELearning, which groups participants and uses intermediate aggregation results for backdoor detection. However, their robustness algorithm is ineffective in detecting poisoning attacks during early training rounds.

In this study, we introduce ClusterGuard, a novel federated learning scheme to address both secure aggregation and poisoning attack problems. It ensures secure client clustering and robustness against adversaries through a combination of advanced cryptographic methods. ClusterGuard also includes a dual-threshold filtering algorithm to identify and exclude malicious clusters. We provide two variants for deployment in both client-server and decentralized environments supported by blockchain. A comprehensive comparison of ClusterGuard with related schemes in terms of complexity and properties is provided in Table I.The contributions of this paper are as follows:

- We propose ClusterGuard, a robust and privacy-preserving federated learning scheme that employs Verifiable Random Functions (VRF) for secure client clustering, Lattice-based Key-Homomorphic Pseudorandom Functions (LKH-PRF) and Verifiable Secret Sharing (VSS) for secure aggregation, and a dual-threshold filtering algorithm for detecting and excluding malicious clusters. We also provide two variants of the scheme tailored for centralized (client-server) and decentralized settings.
- We conduct a rigorous security and theoretical analysis, accompanied by detailed performance assessments, to validate our scheme's theoretical security guarantees and feasibility.
- Extensive experiments on common datasets demonstrate over $2\times$ performance improvement compared to advanced secure aggregation methods. Additionally, under attacks from 20% of malicious clients, the training model maintains a high level of accuracy comparable to the original model, surpassing current state-of-the-art robustness solutions.

II. Models and Preliminaries

Before introducing the models and preliminaries, we list the notations used in our scheme in Table II.

A. Federated Learning

Federated learning (FL) is a distributed training framework where multiple clients collaboratively train a machine learning model. The classic FL algorithm, FedAvg [14], involves n clients and a server. Each client $u_i \in \mathcal{U}$ holds its local dataset $\mathcal{D}_i$. The server initializes the global model parameter θ_0 and broadcasts it to all clients. Participants repeat the following steps for each round r until convergence:

Step 1. Each client receives the global model θ_r and calculates local model updates $\mathbf{w}^{(u_i)} = \nabla F_i(\theta_r, \xi_i)$ by performing multiple steps of Stochastic Gradient Descent (SGD) on the local available data ($\xi_i \sim \mathcal{D}_i$), where F_i is a loss function.

Step 2. Clients upload weight updates $\mathbf{w}_r^{(u_i)}$ to the server.

Step 3. The server aggregates updates and sends back $\theta_{r+1} \leftarrow \theta_r - \eta \frac{1}{n} \sum_{u_i \in \mathcal{U}} \mathbf{w}_r^{(u_i)}$, where η is the learning rate.

B. Verifiable Secret Sharing

Based on Feldman's VSS [15] , Verifiable Secret Sharing (VSS) enables secure distribution of a secret among multiple parties without a dealer. It ensures share integrity through commitments to polynomial coefficients, facilitating secret reconstruction by collaborating participants.

- $\mathbf{VSS.Setup}(1^\lambda) \rightarrow (G, q, g)$, where g is a generator of group G with prime order q.
- $\mathbf{VSS.Share}(s_u, t, \mathcal{U}) \rightarrow (\{v, s_{u,v}\}_{v\in\mathcal{U}}, \{g^{s_{u,v}}\}_{v\in\mathcal{U}}, \{g^{a_{u,d}}\}_{d\in[0,t-1]})$. On input $s_u \in \mathbb{Z}_q$, client u chooses $a_{u,1}, \cdots, a_{u,t-1} \xleftarrow{\$} \mathbb{Z}_q$, constructs a polynomial $f_u(x) = a_{u,0} + a_{u,1}x + \ldots + a_{u,t-1}x^{t-1}$ where $a_{u,0} = s_u$ and $a_{u,t-1} \neq 0$, computes $\{g^{a_{u,d}}\}$ for $\forall d \in [0, t-1]$, $s_{u,v} := f_u(v)$ and $g^{s_{u,v}}$ for $\forall v \in [1, |\mathcal{U}|]$.
- $\mathbf{VSS.Verify}(v, g^{s_{u,v}}, \{g^{a_{u,d}}\}_{d\in[0,t-1]}) \rightarrow b$. This algorithm outputs 1 if and only if the following equation holds:
$$g^{s_{u,v}} \stackrel{?}{=} \prod_{d=0}^{t-1} (g^{a_{u,d}})^{v^d} \mod q$$
- $\mathbf{VSS.Accum}(\{v, s_{u,v}\}_{u\in\mathcal{U}}) \rightarrow (v, s_v^*)$. On input $|\mathcal{U}|$ inputs of the form $(v, s_{u,v})_{u\in\mathcal{U}}$, this algorithm enables client v to accumulate $|\mathcal{U}|$ shares by computing $s_v^* = \sum_{u\in\mathcal{U}} s_{u,v}$ and $g^{s_v^*}$.
- $\mathbf{VSS.VrfyAccum}(v, g^{s_v^*}, \{g^{s_{u,v}}\}_{u\in\mathcal{U}}) \rightarrow b$. This algorithm outputs 1 if and only if the following equation holds: $g^{s_v^*} \stackrel{?}{=} \prod_{u\in\mathcal{U}} g^{s_{u,v}} \mod q$.
- $\mathbf{VSS.Recon}(\{v, s_v^*\}_{v\in\mathcal{U}}) \rightarrow s$. On input at least t inputs of the form (v, s_v^*), this algorithm computes and outputs $s = f(0) = \sum_{u\in\mathcal{U}} s_u$.

C. Lattice-based Key-homomorphic PRF

Given spaces $\mathcal{X}, \mathcal{Y}, \mathcal{K}$ over $\{0,1\}^*$, let $(\mathcal{K}, \star)$ and $(\mathcal{Y}, \bullet)$ be groups and have a group homomorphism between the space $\mathcal{K}$ and $\mathcal{Y}$ [16]. For $\forall x \in \mathcal{X}$ and $k_1, k_2 \in \mathcal{K}$, a pseudorandom function $\text{PRF}_k : \mathcal{X} \rightarrow \mathcal{Y}$ where $k \in \mathcal{K}$ denotes the key is key-homomorphic, if the following equation holds: $\text{PRF}_{k_1}(x) \bullet \text{PRF}_{k_2}(x) = \text{PRF}_{k_1 \star k_2}(x)$.

The above properties can be realized by the lattice cryptography. In fact, the lattice-based KH-PRF is an approximate homomorphic function, formulated as [17]: $\text{LKH-PRF}_{k_1+k_2}(x) = \text{LKH-PRF}_{k_1}(x) + \text{LKH-PRF}_{k_2}(x) + e$, where $e \in \mathbb{N}$ is small. Among existing lattice-based KH-PRF schemes, Boneh et al.' s construction [17] based on LWR is simpler and more efficient, and its security is based on ROM. Let $H : \mathcal{X} \rightarrow \mathbb{Z}_\mathfrak{q}^\ell$ be a hash function modeled as a random oracle, $\mathbf{k} \in \mathbb{Z}_\mathfrak{q}^\ell$ be a vector key, the lattice-based KH-PRF is defined as follows:

$$\text{LKH-PRF}_\mathbf{k}(x) = \lceil \langle H(x), \mathbf{k} \rangle \rfloor_\mathfrak{p} \tag{2}$$

TABLE I
COMMUNICATION AND COMPUTATION OVERHEAD COMPARED WITH OTHER SECURE AGGREGATION SCHEMES.

Setting	Bonawitz [2]	Bell [3]	FLDP [13]	ACORN [7]	LERNA [10]	Ours
Client Communication	$O(n+m)$	$O(\log n+m)$	$O(m+k'+n)$	$O(m+\log n)$	$O(m)$	$O(\ell l+m+n)$
Client Computation	$O(n^2+nm)$	$O(\log^2 n+m\log n)$	$O(mk'+n\log n)$	$O(m\log n)$	$O(mlog\ell+M\ell)$	$O(\ell l^2+m\ell+n)$
Server Communication	$O(n^2+nm)$	$O(n\log n+nm)$	$O(mn+k')$	$O(nm+n\log n)$	$O((M+n)m)$	$O(mn+\ell n)$
Server Computation	$O(mn^2)$	$O(n\log^2 n+nm\log n)$	$O(mk'+mn+n\log n)$	$O(nm\log n)$	$O(Mt)$	$O(nt+\frac{nm\ell}{l})$
Pairwise Masking	Yes	Yes	No	Yes	No	No
Mask Error	No	No	Yes	No	No	No
Clustering Mechanism	✗	✓	✗	✓	✗	✓
Robustness Mechanism	No	No	No	Yes	No	Yes

* n is the number of clients, m is the input size or the model vector size, k' is the key size in FLDP [13], ℓ is the key size of our scheme, and t is the threshold of secret sharing, l is the number of members in each cluster, M is the commitee size of LERNA. For fair comparison, the computation and communication of LERNA is for non-member clients (including flat secret sharing).

TABLE II
NOTATIONS

$\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{k}, \mathbf{s}, \mathbf{s}_u$	Vector format
s, s_u, a_i	Scalar format
$\langle \mathbf{x} \cdot \mathbf{y} \rangle$	Inner product
$\{s_u\}_{u\in\mathcal{U}}$	A set of scalar values
$\mathcal{U} = \{u_1, u_2, \cdots, u_n\}$	A set of clients
$\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots \cup \mathcal{C}_k\}$	Clients split into clusters
k	the number of clusters
l	the number of members in each cluster
n	the number of clients

where $\lceil x \rfloor_{\mathfrak{p}}$ denotes the rounding operation as $\lceil x \rfloor_{\mathfrak{p}} = \lfloor x \cdot (\mathfrak{p}/\mathfrak{q}) \rfloor \mod \mathfrak{p}$. Ernst et al. [16] complete the security proof of the LKH-PRF, provide a simple construction of $H(x)$, and introduce a rounding error $e \in \{0, 1\}$ in their construction. Let $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ where $x_i \in \mathcal{X}$, we can extend LKH-PRF for m-dimensional vectors as follows: $\text{LKH-PRF}_{\mathbf{k}}(\mathbf{x}) = [\text{LKH-PRF}_{\mathbf{k}}(x_1), \cdots, \text{LKH-PRF}_{\mathbf{k}}(x_m)]$.

D. Verifiable Random Function

Verifiable Random Functions (VRFs) [18] are cryptographic primitives that extend the concept of random functions by enabling third parties to verify the correctness of the outputs without knowing the secret keys. A Formal Definition is as follows:

- $\mathsf{KeyGen}(1^\lambda) \to (sk_u, pk_u)$ Generate the corresponding key pair (sk_u, pk_u) based on the public security parameter λ.
- $\mathsf{Eval}(sk_u, m_u) \to (r_u, \pi_u)$ Given the secret key sk_u, an input $m_u \in \{0,1\}^*$, the evaluation algorithm outputs random value $r_u \in \mathbb{N}$ along with a proof π_u, such that the correctness of the output can be verified by pk_u.
- $\mathsf{Check}(pk_u, m_u, r_u, \pi_u) \to \{0, 1\}$ Given pk_u, m_u, r_u, and π_u, the verification algorithm outputs 1 if r_u is correct, and 0 otherwise.

E. Local Outlier Factor Algorithm

The Local Outlier Factor (LOF) algorithm [19] quantifies the degree of abnormality of a data point by comparing the local density of the point with that of its neighbors. The main steps are as follows:

- For a given data point p, use the K-Nearest Neighbors algorithm [20] to find its K nearest neighbors $N_K(p)$.
- For two data points p and o, the reachability distance is defined as:$\text{reach-dist}_K(p, o) = \max(d_K(o), d(p, o))$, where $d_K(o)$ is the distance from o to its K-th nearest neighbor, and $d(p, o)$ is the distance between p and o.
- The LRD of a point p is computed as:

$$\text{LRD}_K(p) = \frac{1}{\frac{1}{|N_K(p)|}\sum_{o\in N_K(p)} \text{reach-dist}_K(p, o)} \quad (3)$$

  This represents how densely p is surrounded by its neighbors. Lower LRD implies potential anomaly.
- The LOF score of point p is:

$$\text{LOF}_K(p) = \frac{\sum_{o\in N_K(p)} \text{LRD}_K(o)}{|N_K(p)| \cdot \text{LRD}_K(p)} \quad (4)$$

A higher LOF score indicates that p is significantly less dense than its neighbors, suggesting it may be an outlier. Anomaly detection is performed using a threshold $\tau > 1$:

$$O_K(p) = \begin{cases} 1 & \text{if LOF}_K(p) > \tau \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

III. RELATED WORK

A. Secure aggregation based on MPC

MPC enables joint computation over private inputs, suitable for secure federated learning aggregation. SecureML [21] employs two-party computation for privacy, but only for basic neural networks. Google's scheme [2] utilizes double masking and secret sharing with high computational overhead. New approaches like FastSecAgg [4] and Laf [22] optimize secret sharing but overlook double mask complexity. CCESA [23], SecAgg+ [3], and Turbo-Aggregate [5] alter client communication to reduce complexity. However, they still rely on double masks, leading to increased computation with dropped clients. As dropped clients rise, both server and client computation and communication increase significantly. Liu et al. [8] used homomorphic PRG based on Diffie-Hellman assumption to construct a single mask protocol for the secure aggregation, but the whole protocol requires additional discrete logarithm calculation when recovering model aggregation, which hinders its application in the actual scenario. SASH [9] and LERNA [10] proposed using (R)LWR to construct homomorphic masks for single-server aggregation, which is similar to our paper, but our scheme requires fewer public parameters. SASH did

not consider how to solve the error problem caused by homomorphic masks. And LERNA uses large-scale heavy clients to serve as committee members to address errors and mask reconstruction issues, which increases the actual deployment difficulty of the scheme. At the same time, their aggregation scheme cannot provide security against poisoning attacks.

B. Robust aggregation of federated learning

Poisoning attacks, divided into data and model poisoning, pose threats to machine learning models. Data poisoning involves adding misleading samples to training data, leading to incorrect predictions [24]. Model poisoning entails sending false updates to global models, impacting performance in distributed settings like federated learning [25], [26]. [27]–[31] proposed defenses using Euclidean distance, median, and cosine similarity. FedDMC [32]and RoPE [33] utilize dimensionality reduction through Principal Component Analysis (PCA) combined with clustering algorithms to detect poisoning attacks and provide defense. But they require servers to access plaintext updates, risking privacy. SecureFL [34], PEFL [35], FLOD [36], PBFL [37], ShieldFL [38], and RFed [39] enable secure computation of similar metrics but rely on two or more non-colluding servers to ensure privacy, which is challenging in practice. Robust aggregation schemes have also been proposed for single-server scenarios. Rofl [11] and ACORN [7] employ zero-knowledge proofs to achieve privacy protection and imposes norm constraints on model updates. However, these schemes incur significant additional computational overhead, especially when clients generate the proofs. Hao et al. [40] use multi-key homomorphic encryption to compute distances between model updates for anomaly detection, but it causes large ciphertext expansion and high communication overhead due to frequent ciphertext broadcasts for client-assisted decryption. Cluster aggregation addresses these challenges, allowing the server to aggregate cluster values without revealing individual updates. Several related works [41]–[43] explore this approach, but these studies either require the server has a certain scale of verification data set, or can not guarantee that the performance of the model is not damaged. FedGT [44] partitions clients into overlapping test groups based on an assignment matrix and uses a validation set to evaluate group-level models via principal components, then applies probabilistic modeling to identify malicious clients. However, the experimental scale is limited to only $15 \sim 30$ clients, and the complexity of the scheme restricts its applicability in large-scale federated learning scenarios.

IV. Our Scheme

A. Threat Model

We consider two different types of attackers, consistent with SAFELearning [12]. The first type of attacker aims to steal the privacy of model updates from honest clients, while the second type seeks to disrupt the normal training of federated learning. We assume that the server and the second type of attackers are adversarial, as the server may also benefit from an accurate and robust global model.

1) Type I attackers: Potential type I attackers include the server and some clients. It is important to note that collusion may occur both between clients and between the server and clients, with the aim of compromising the model privacy of honest clients. Clients may be malicious, while the server is assumed to be honest but curious.
2) Type II attackers: For type II attackers, the attackers consist of one or more malicious clients, who can upload arbitrary model updates to perform poisoning attacks, with the goal of disrupting the training of the entire global model. The attackers cannot influence the behavior of benign users, nor can they observe the local models of benign users. In the cluster aggregation scheme, we assume that the proportion of Type II attackers (malicious or compromised users) is less than $\frac{k}{n}$ to ensure that at least one cluster consists entirely of benign clients. Additionally, we assume that the attackers can fully cooperate with each other.

B. Overview

We propose a new secure aggregation framework to address the dual challenges of protecting model update privacy and detecting malicious updates. Clients are divided into clusters prior to secure aggregation, with secure aggregation executed within each cluster. Aggregation from each cluster can then be analyzed to identify potential malicious updates.

However, in clustered federated learning settings, collusion attacks involving the server and malicious clients pose a significant threat. By manipulating cluster assignments, adversaries could concentrate a large number of malicious clients in a single cluster, compromising the privacy of honest clients within that cluster. To counter this, we employ verifiable random numbers. Each client u_i generates a random number r_i along with a corresponding proof π_i based on a message m_i, which are broadcast to other clients. Only after receiving all random numbers can clients determine their respective cluster memberships.

To safeguard client model privacy, we design a secure aggregation protocol leveraging lattice-based key-homomorphic pseudorandom functions (LKH-PRF) and verifiable secret sharing (VSS). For each cluster $\mathcal{C}_\varphi$, each client independently generates its secret key vector $\mathbf{s}_u$ and shares it with other cluster members. Cluster members then forward their received key vector shares to the server. The server reconstructs $\mathbf{s} = \sum \mathbf{s}_u$ and performs unmasking operations using:

$$\text{LKH-PRF}_{\mathbf{s}}(\mathbf{x}) = \sum_{u \in \mathcal{C}_\varphi} \text{LKH-PRF}_{\mathbf{s}_u}(\mathbf{x}).$$

This clustered architecture significantly reduces communication complexity from $O(n^2)$ to $O(l^2)$. Moreover, the error introduced by LKH-PRF is effectively controlled.

After receiving aggregated updates from all clusters, the server applies a dual-threshold filtering mechanism based on norms and cosine similarity to detect potential poisoning attacks. The server maintains a small trusted dataset locally to constrain the norm and cosine similarity of cluster updates

within tolerable ranges. This adaptive mechanism dynamically adjusts the tolerance thresholds to counter different attack scenarios. Additionally, to safeguard global model performance, we implement a "baffle" mechanism to mitigate extreme attack cases.

To accommodate real-world applications, we propose two variants of our scheme: one tailored to the single server-client model and another designed for decentralized settings. For consistency and clarity, the main description focuses on the single server-client model, while the decentralized variant is presented as an extension in the discussion section. The overall architecture of our solution is illustrated in Figure 1, which outlines the four-round interaction process.

C. VRF Based Clustering

We employs Verifiable Random Function (VRF) to facilitate client clustering and assignment. Each client generates a random value and proof using its private key, which is then broadcasted to other clients. Upon receiving the broadcasts, clients perform batch verification. If the verification is successful, the clustering of clients is determined based on the aggregated randomness. The steps involved in this process are as follows:

Consider a bilinear pairing $e : G_1 \times G_2 \to G_T$, where G_1, G_2 and G_T are multiplicative cyclic groups of prime order p, and g_1, g_2 are the generators of G_1 and G_2 respectively. Let $H_1 : \{0,1\}^* \to G_1$ be a public hash function that maps an arbitrary-length binary message m to an element in the group G_1. Additionally, let $H_2 : G_1 \to \mathbb{N}$ be a public, deterministic function that maps an element of the group G_1 to an integer in $\mathbb{N}$.

*Step 1. Key Generation (*KeyGen*):* Each client generates a key pair (sk_i, pk_i), where the private key sk_i is generated using a secure key generation algorithm. The corresponding public key pk_i is computed as $pk_i = g_2^{sk_i}$. Afterward, each client broadcasts its public key pk_i to all other clients.

*Step 2. VRF Generation (*Eval*):* Each client u_i first generates a random byte string $m_i \in \{0,1\}^*$, then computes its proof as $\pi_i = H_1(m_i)^{sk_i}$. The corresponding random value r_i is derived as $r_i = H_2(\pi_i)$. The proof π_i, along with the values (r_i, m_i), is then broadcast to other clients and the server.

*Step 3. Batch verification (*Check*):* After receiving all the broadcasted values (m_i, r_i, π_i), each client verifies their validity by utilizing the corresponding public keys pk_i and proofs π_i. The verification is conducted using the BLS aggregation verification [45]:

$$\begin{aligned} \pi &= \prod_{i=1}^{n} \pi_i = H_1(m_1)^{sk_1} \cdot H_1(m_2)^{sk_2} \cdots H_1(m_n)^{sk_n} \\ e(\pi, g_2) &\stackrel{?}{=} \prod_{i=1}^{n} e(H_1(m_i), pk_i), \quad \bigwedge_{i=1}^{n} (r_i = H_2(\pi_i)) \stackrel{?}{=} 1 \end{aligned} \tag{6}$$

where $\bigwedge$ represents the logical AND operation. If the verification is successful, the client continues with the protocol.

Step 4. Aggregation of Randomness: All clients compute the aggregated randomness R_{total} by performing a bitwise XOR operation: $R_{\text{total}} = \bigoplus_{i=1}^{n} r_i$, where $\bigoplus$ represents the bitwise XOR operation.

Step 5. Cluster Assignment: Each client computes its cluster index φ_i based on the aggregated randomness R_{total}, the current federated learning round r and a uniform hash function:

$$\varphi_i = \text{Hash}(i \| R_{\text{total}} \| r) \mod k \tag{7}$$

where φ_i denotes the cluster $\mathcal{C}_\varphi$ to which client u_i is assigned, and r corresponds to the current round of the federated learning process. The secure aggregation protocol is then executed within each cluster to collect model updates.

D. LKH-PRF Model Mask and Aggregation

In our scheme, each user $u \in \mathcal{U}$ holds the m-dimensional model update $\mathbf{x}_u \in \mathbb{R}^m$, while the outputs of LKH-PRF are elements of $\mathbb{Z}_{\mathfrak{p}}$, thus we define an encoding scheme $(\text{Encode}, \text{Decode})$ as follows:

$$\begin{aligned} &\text{Encode} : \mathbb{R}^m \to \mathbb{Z}^m : \mathbf{x}_u \mapsto \hat{\mathbf{x}_u} = \lfloor (\mathbf{x}_u + \mathbf{b}) \cdot 2^{prec} \rfloor \\ &\text{Decode} : \mathbb{Z}^m \to \mathbb{R}^m : \hat{\mathbf{x}_u} \mapsto \mathbf{x}_u = \frac{\hat{\mathbf{x}_u}}{2^{prec}} - \mathbf{b} \end{aligned} \tag{8}$$

where $prec$ is the number of bits of precision and $\mathbf{b}$ is a constant offset to ensure encoding as unsigned integers. Note that scalar multiplication, modulo and comparison operations on vectors are applied individually each of their elements.

As described in Equation (II-C), let $\{\ell, \mathfrak{p}, \mathfrak{q}, \mathbf{c}\}$ be the parameters of $\text{LKH-PRF}_{\mathbf{k}}$ where PRF is basd on $\text{LWR}_{\ell,\mathfrak{q},\mathfrak{p}}$, $\mathbf{k} \in \mathbb{Z}_{\mathfrak{q}}^{\ell}$ and $\mathbf{c} \in \mathbb{Z}_{\mathfrak{q}}^{m}$ is a non-repetitive sequence representing label values for various parameters of the global model, e.g., $\mathbf{c} = (c||1, c||2, \cdots, c||m)$ and $c \in \mathbb{Z}_{\mathfrak{q}}$. In the setup phase, the public parameter $\mathbf{c}$ only requires an additional scalar $c \in \mathbb{Z}_q$, rendering it simpler compared to the matrix construction [9], [10]. Thus, the public parameters are $\{\ell, \mathfrak{p}, \mathfrak{q}, c\}$.

Consider a subset $\mathcal{U}'$ of $\mathcal{U}$. Each client $u \in \mathcal{U}'$ holds a secret key $\mathbf{s_u}$, and it can compute its masked model update as follows:

$$\mathbf{y}_u = \Delta \cdot \text{Encode}(\mathbf{x}_u) + \text{LKH-PRF}_{\mathbf{s}_u}(\mathbf{c}) \tag{9}$$

where $\Delta > |\mathcal{U}'|$ is a scaling factor. The server holds the secret key $\mathbf{s} = \sum_{u \in \mathcal{U}'} \mathbf{s_u}$ constructed with key shares and computes the aggregate masks as follows:

$$\text{LKH-PRF}_{\mathbf{s}}(\mathbf{c}) = \sum_{u \in \mathcal{U}'} \text{LKH-PRF}_{\mathbf{s}_u}(\mathbf{c}) + \mathbf{e} \mod \mathfrak{p} \tag{10}$$

where $\mathbf{e} = (e_1, \ldots, e_m)$ is a vector composed of errors and $0 \le e_i < |\mathcal{U}'|$ since the summation of LKH-PRF causes the accumulation of its error from $\{0, 1\}$. With reasonable parameter selection, the secure aggregation can be done as follows:

$$\begin{aligned} \hat{\mathbf{z}} &= \frac{1}{|\mathcal{U}'|\Delta} \Big(\sum_{u \in \mathcal{U}'} \mathbf{y}_u - \text{LKH-PRF}_{\mathbf{s}}(\mathbf{c}) \Big) \\ &= \frac{1}{|\mathcal{U}'|\Delta} \sum_{u \in \mathcal{U}'} \Delta \cdot \hat{\mathbf{x}_u} - \frac{\mathbf{e}}{\Delta} \mod \mathfrak{p} \\ &= \sum_{u \in \mathcal{U}'} \frac{1}{|\mathcal{U}'|} \cdot \text{Encode}(\mathbf{x}_u) \end{aligned} \tag{11}$$

It can be seen that the solution to the error introduces a scaling factor. Although the error issue is resolved, a new problem arises: the scaling factor must be at least greater than the number of clients. As the number of clients increases, for a

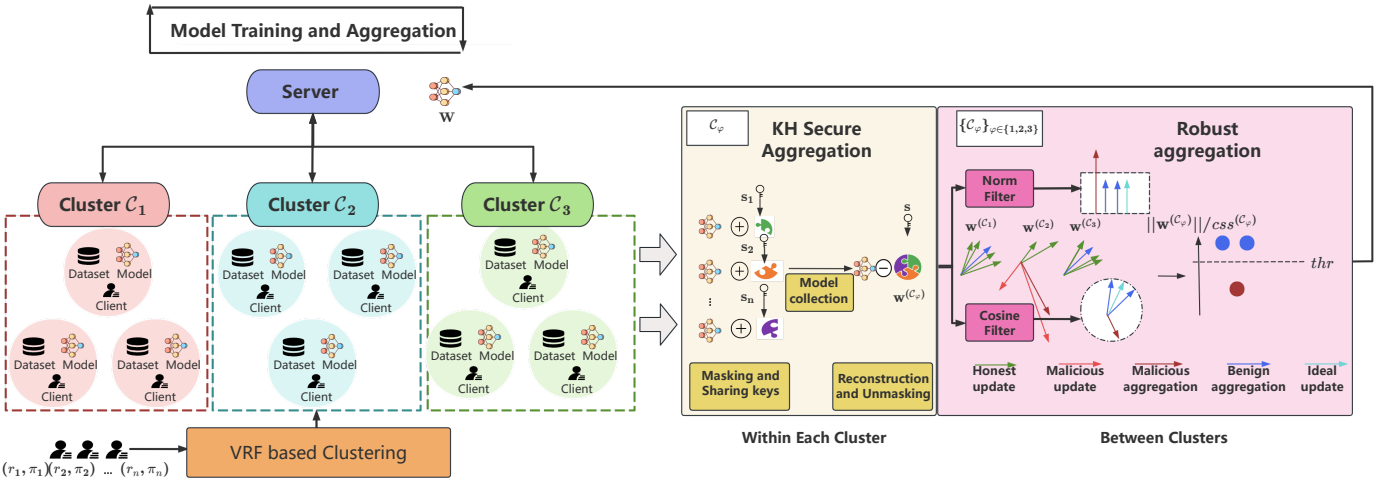


Fig. 1. The architecture of ClusterGuard. Initially, the clients are divided into clusters, with the clustering determined based on the VRFs, and secure aggregation is performed within each cluster. Each client calculates the corresponding model mask according to its secret key $\mathbf{s_i}$, and uploads the masked model. The server directly calculates the model mask and eliminates the secure aggregation model according to the total secret key $\mathbf{s}$. After conducting robust aggregation, the server obtains the correct aggregated model update.

fixed modulus $\mathfrak{p}$, this will result in a decrease in the maximum number of bits that the target vector can add a mask to. Since the server will compute $\sum_{u\in\mathcal{U}'} \Delta \cdot \text{Encode}(\mathbf{x}_u)$, and it is necessary to ensure no overflow occurs, the number of bits that each client can use to transmit its original data does not exceed $\lfloor \log_2(\frac{\mathfrak{p}}{|\mathcal{U}'|^2}) \rfloor$. Therefore, when the number of clients is large and high model accuracy is required (i.e., more bits need to be transmitted), it becomes necessary to limit the number of model aggregation operations. A fixed cluster size $|\mathcal{U}'| = |\mathcal{C}_\varphi| = l$ can meet this requirement. After decoding the aggregated result, the final aggregated model update can be computed as:

$$\mathbf{z} = \text{Decode}(\hat{\mathbf{z}}) = \text{Decode}(\sum_{u\in\mathcal{U}'} \frac{1}{|\mathcal{U}'|}\hat{\mathbf{x}_u}) = \frac{1}{|\mathcal{U}'|}\sum_{u\in\mathcal{U}'} \mathbf{x}_u \tag{12}$$

E. Cluster-based Secure Aggregation

We use verifiable secret sharing to enable the server to obtain the sum of the clients' secret keys. Based on this, we construct our cluster-based secure aggregation scheme.

Initialization. All clients are randomly partitioned into k clusters, i.e., $\mathcal{U} = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \ldots \cup \mathcal{C}_k$ ($l = |\mathcal{C}_\varphi|$ for $\varphi \in [1, k]$). The public parameters are generated using $\mathbf{VSS.Setup}$, with a secret sharing threshold of t. Each client has completed local training for the current round of federated learning and holds a model update $\mathbf{x}_u$. For each cluster $\mathcal{C}_\varphi \subset \mathcal{U}$, every client $u \in \mathcal{C}_\varphi$ selects a random ℓ-dimensional vector key $\mathbf{s}_u$.

Key Sharing. Each client $u \in \mathcal{C}_\varphi$ utilizes $\mathbf{VSS.Share}$ to generate l secret shares $\{v, s_{u,v}\}_{v\in\mathcal{U}}$ for its key $\mathbf{s}_u$, and distributes these shares to other clients $v \in \mathcal{C}_\varphi$.

Model update collection. Each client $u \in \mathcal{C}_\varphi$ computes masked model update $\mathbf{y}_u$ using $\mathbf{s}_u$, and uploads the masked update $\mathbf{y}_u$ to the server. After collecting all the model updates, the server notifies each client $u \in \mathcal{C}_\varphi$.

Unmasking. Each client $u \in \mathcal{C}_\varphi$ utilizes $\mathbf{VSS.Accum}$ to accumulate the secret shares (u, s_u^*) received from other clients v, and uploads the accumulated share to the server. For each cluster $\mathcal{C}_\varphi \subset \mathcal{U}$, the server collects and verifies at least t accumulated shares from clients $u \in \mathcal{C}_\varphi$, and then invokes $\mathbf{VSS.Recon}$ to reconstruct the aggregated cluster key $\mathbf{s}_{\mathcal{C}_\varphi} = \sum_{u\in\mathcal{C}_\varphi} \mathbf{s}_u$. The server can then perform unmasking and decoding to compute the aggregated model update $\hat{\mathbf{z}}$ using one of the following two strategies:

- Strategy I (without considering robust aggregation): Aggregate all cluster keys to compute the global key $\mathbf{s} = \sum_{\mathcal{C}_\varphi\subset\mathcal{U}} \mathbf{s}_{\mathcal{C}_\varphi}$, and then aggregate model updates across all clusters. In this case, Δ must satisfy $\Delta > n$, and the aggregated model is computed as:

$$\hat{\mathbf{z}} = \frac{1}{n\Delta}(\sum_{u\in\mathcal{U}} \mathbf{y}_u - \text{LKH-PRF}_{\mathbf{s}}(\mathbf{c})) \tag{13}$$

- Strategy II (considering robust aggregation): First aggregate model updates within each cluster, then aggregate across all clusters. In this case, Δ must satisfy $\Delta > l$, and the aggregated model is computed as:

$$\hat{\mathbf{z}} = \frac{1}{k}\sum_{\mathcal{C}_\varphi\subset\mathcal{U}} \left(\frac{1}{l\Delta}(\sum_{u\in\mathcal{C}_\varphi} \mathbf{y}_u - \text{LKH-PRF}_{\mathbf{s}_{\mathcal{C}_\varphi}}(\mathbf{c})) \right) \tag{14}$$

Handling Dropped Clients. In the setting of cluster-based secure aggregation, client dropouts may hinder the removal of masking. To address this issue, the following mechanisms are designed:

- Case I: For each cluster $\mathcal{C}_\varphi$, some clients may drop out during model update collection. The server records the subset of clients $\mathcal{C}'_\varphi$ that successfully upload, then returns $\mathcal{C}'_\varphi$ to all clients. Each client $u \in \mathcal{C}'_\varphi$ uses this set to accumulate the secret shares $s_{v,u}$ received from $v \in \mathcal{C}'_\varphi$, and sends the result to the server.
- Case II: Each client $u \in \mathcal{C}'_\varphi$ may also drop out during the upload of its accumulated secret share in the unmasking phase. However, leveraging verifiable secret sharing, the

server only needs to wait for and verify at least t accumulated shares to reconstruct the sum of the secret keys from the $|\mathcal{C}'_\varphi|$ participating clients, i.e., $\mathbf{s}_{\mathcal{C}'_\varphi} = \sum_{u\in\mathcal{C}'_\varphi} \mathbf{s}_u$, thereby enabling mask removal.

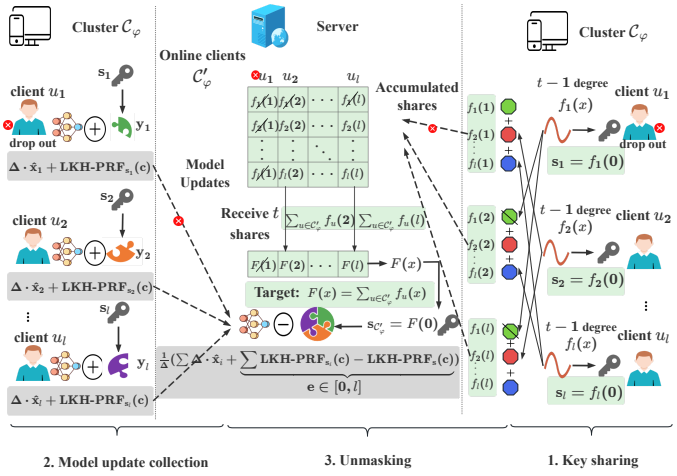


Fig. 2. Each cluster performs secret sharing internally. After collecting all masked model updates, the server notifies each client $u \in \mathcal{U}$ that needs to calculate the sum of secret shares (i.e., all clients that have successfully uploaded masked model updates in a cluster). Then the server only needs to wait for t clients within the cluster to upload the sum of shares to reconstruct the cluster's total secret key $\mathbf{s}$.

F. Robust Aggregation

It can be observed that the server can obtain the aggregated model updates of the clients within the cluster but cannot access individual model updates, enabling robust detection while preserving privacy. Similar to FLtrust [30], [37], the server collects a small, clean root dataset to compute the ideal model update. The similarity between the ideal update and each cluster's aggregated update is used to assess aggregation safety. Under the cluster collection, the cluster is divided into three situations: (1) all benign clients, (2) mixed malicious clients and benign clients, and (3) all malicious clients. An adaptive dual-threshold mechanism is proposed as follows:

Cosine Similarity Threshold Filtering. Let $\mathbf{w}_r^{(\mathcal{C}_\varphi)}$ ($\varphi \in [1, k]$) denote the aggregated update of cluster $\mathcal{C}_\varphi$ at round r, and g_r^* be the ideal (trusted) update. The cosine similarity score is defined as:

$$css_r^{(\mathcal{C}_\varphi)} = \frac{\langle g_r^*, \mathbf{w}_r^{(\mathcal{C}_\varphi)} \rangle}{\|g_r^*\|_2 \cdot \|\mathbf{w}_r^{(\mathcal{C}_\varphi)}\|_2} \tag{15}$$

Clusters with similarity scores below a threshold thr_1 are considered malicious and excluded from the final aggregation. The threshold thr_1 is adaptively computed based on a hyperparameter ρ, allowing tolerance for deviations from honest clients. The filtered cluster set is:

$$\mathcal{U}'_1 \leftarrow \{\mathcal{C}_\varphi \subset \mathcal{U} \mid css_r^{(\mathcal{C}_\varphi)} > thr_1\} \tag{16}$$

Norm Threshold Filtering. Let the norm distance score be defined as:

$$ndis_r^{(\mathcal{C}_\varphi)} = \left| \frac{\|\mathbf{w}_r^{(\mathcal{C}_\varphi)}\|_\infty}{\|g_r^*\|_\infty} - 1 \right| \tag{17}$$

If the norm of a cluster update deviates significantly from the ideal update, i.e., $ndis > thr_2$, the cluster is considered malicious. The filtered cluster set becomes:

$$\mathcal{U}'_2 \leftarrow \{\mathcal{C}_\varphi \subset \mathcal{U} \mid ndis_r^{(\mathcal{C}_\varphi)} < thr_2\} \tag{18}$$

Combining both filters, the final robust cluster set and aggregated model update are:

$$\mathbf{z}' = \frac{1}{k'} \sum_{\mathcal{C}_\varphi \subset \mathcal{U}'} \mathbf{w}_r^{(\mathcal{C}_\varphi)} = \frac{1}{k'l} \sum_{u\in\mathcal{U}'} \mathbf{w}_r^{(u)} \tag{19}$$

where $\mathcal{U}' = \mathcal{U}'_1 \cap \mathcal{U}'_2$ and k' is the number of clusters in $\mathcal{U}'$.

To prevent degradation under extreme conditions (e.g., when all clusters deviate significantly from the trusted update), a blocking mechanism is introduced. If all similarity scores fall below hard thresholds bl_1 or bl_2, all updates are rejected to preserve model integrity. Let **Max**$(\cdot)$ denote the maximum, **Min**$(\cdot)$ the minimum, and **Stdev**$(\cdot)$ the standard deviation function. The robust aggregation procedure is detailed in Algorithm 1.

Algorithm 1 RobustAgg

Input: Global model θ_r, model update aggregations of all clusters $\mathcal{Q} = \{\mathbf{w}_r^{(\mathcal{C}_1)}, \mathbf{w}_r^{(\mathcal{C}_2)}, \cdots, \mathbf{w}_r^{(\mathcal{C}_k)}\}$ (recover the model structure from the aggregated vectors $\{\mathbf{z}_{\mathcal{C}_\varphi}\}_{\mathcal{C}_\varphi\subset\mathcal{U}}$), root data set $\mathcal{D}_0$, local learning rate η.

Output: Robustness aggregation $\mathbf{w}_r$

Initialization: Generate $dis_1 = dis_2 = \emptyset$, $\mathcal{Q}'_1 = \mathcal{Q}'_2 = \emptyset$, the weight factor $\rho > 1$
Step 1: Trusted update calculation
Sample the root dataset for stochastic gradient descent to obtain the trusted model update g_r^*.
Step 2: Norm-based Filtering
for each cluster $\mathcal{C}_\varphi$ **do**
  $ndis_r^{(\mathcal{C}_\varphi)} \leftarrow |\frac{\|\mathbf{w}_r^{(\mathcal{C}_\varphi)}\|_\infty}{\|\mathbf{g}_r^*\|_\infty} - 1|$
  $dis_1 \leftarrow dis_1 \cup ndis_r^{(\mathcal{C}_\varphi)}$
$thr_1 \leftarrow$ **Min**$(dis_1) \cdot \rho$
if **Min**$(dis_1) > bl_1$:
  $\mathcal{Q}'_1 = \emptyset$
else:
  $\mathcal{Q}'_1 = \{\mathbf{w}_r^{(\mathcal{C}_\varphi)} | ndis_r^{(\mathcal{C}_\varphi)} < thr_1\}$
Step 3: Cosine-based Filtering
for each cluster $\mathcal{C}_\varphi$ **do**
  $css_r^{(\mathcal{C}_\varphi)} = \frac{\langle g_r^*, \mathbf{w}_r^{(\mathcal{C}_\varphi)} \rangle}{\|g_r^*\| \cdot \|\mathbf{w}_r^{(\mathcal{C}_\varphi)}\|}$
  $dis_2 \leftarrow dis_2 \cup css_r^{(\mathcal{C}_\varphi)}$
$thr_2 \leftarrow$ **Max**$(\{dis_2\}) - \rho \cdot$ **Stdev**(dis_2)
if **Max**$(\{dis_2\}) < bl_2$:
  $\mathcal{Q}'_2 = \emptyset$
else:
  $\mathcal{Q}'_2 = \{\mathbf{w}_r^{(\mathcal{C}_\varphi)} | css_r^{(\mathcal{C}_\varphi)} > thr_2\}$
Step 4: Aggregation
Get trusted set: $\mathcal{Q}' = \mathcal{Q}'_1 \cap \mathcal{Q}'_2$
If $Q' = \emptyset$, then $\mathbf{w}^r$ is a full 0 tensor with the same shape as the original model update, otherwise calculate the aggregate value $\mathbf{w}^r \leftarrow \frac{1}{|\mathcal{Q}'|} \sum \mathcal{Q}'$.
return $\mathbf{w}^r$

G. Multi-round Detection Mechanism

The aforementioned cluster-based robustness algorithm provides security protection for federated learning model training. However, in order to balance privacy protection, it can only detect clusters containing malicious clients, without being able to accurately identify the specific malicious clients. In federated learning systems with strict regulatory requirements, mechanisms need to be designed to accurately pinpoint malicious clients. To address this issue, this section proposes a multi-round detection mechanism that gradually narrows down the identity of potential malicious clients from a multi-round perspective. The core idea is to utilize the robustness algorithm to track the frequency of clusters marked as malicious across rounds and maintain a historical frequency table. Since each round adopts a random cluster generation strategy based on verifiable randomness, honest clients are unlikely to consistently appear in malicious clusters. Therefore, after a certain number of rounds r_t, malicious clients can be identified due to abnormally high marking frequencies. The specific steps are as follows:

Initialization. In the initial stage, set the parameters K and τ for the Local Outlier Factor (LOF) anomaly detection algorithm. A historical frequency table H is initialized for all clients to record how many times each client has been marked as part of a malicious cluster across all rounds. For client $u_i \in \mathcal{U}$, the historical frequency table is initialized as:

$$H_0(u_i) = 0, \ \forall i \in [1, n] \tag{20}$$

Update the historical frequency table. For each federated learning round r, apply the adaptive dual-threshold filtering robustness algorithm to identify the set of safe clusters $\mathcal{U}'$ and update the historical frequency table as follows:

$$H_r(u_i) = \begin{cases} H_{r-1}(u_i) & \text{if } u_i \in \mathcal{U}' \\ H_{r-1}(u_i) + 1 & \text{if } u_i \in \mathcal{U} \setminus \mathcal{U}' \end{cases} \tag{21}$$

Malicious client detection. After a certain number of federated learning rounds r_t, the LOF anomaly detection algorithm can be applied to identify malicious clients. Let $\mathcal{U}_m$ denote the set of detected malicious clients, computed as:

$$\begin{aligned} \bar{H}_{r_t} &\leftarrow \frac{1}{|\mathcal{U}|} \sum_{u_i \in \mathcal{U}} H_{r_t}(u_i) \\ \mathcal{U}_m &\leftarrow \{u_i \mid O_K(H_{r_t}(u_i)) = 1 \text{ and } H_{r_t}(u_i) > \bar{H}_{r_t}\} \end{aligned} \tag{22}$$

where $O_K(\cdot)$ denotes the LOF anomaly detection function under K-nearest neighbors (refer to Equation 5). A client is included in $\mathcal{U}_m$ only if its historical marking frequency is higher than the average $\bar{H}_{r_t}$ and it is detected as an anomaly.

H. Putting it all Together

We give the overall architecture of our solution in Figure 1, which is divided into four rounds. We assume a public key infrastructure where each client holds a key pair $(\mathsf{epk}_u, \mathsf{esk}_u)$ for encryption. We use $\mathsf{Enc}(\mathsf{msg}, \mathsf{epk}_u)$ and $\mathsf{Dec}(\mathsf{msg}, \mathsf{esk}_u)$ to denote encryption and decryption respectively. The specific protocol can be found in Protocol 3.

In **Round 0**, the clients broadcast their VRF values and proofs, which are forwarded by the server. Upon receiving this information, the clients perform verification. Subsequently, the clients are randomly partitioned into multiple clusters. In **Round 1**, each client generates its own secret key and performs secret sharing with other clients within its cluster. In **Round 2**, each client uses the LKH-PRF to compute its masked model update and uploads the result to the server. The server records the online client information for each cluster and sends it back to the clients. Based on the online client list, each client accumulates the received secret shares and uploads the result to the server. The specific operations are shown in Figure 2. In **Round 3**, the server waits to receive a sufficient number of secret shares before beginning to reconstruct the total secret key for each cluster. Subsequently, the server calculates the final model update aggregation for all clients. It is important to note that additional verification operations are required when considering malicious adversaries.

V. Security and Convergence Analysis

In this section, we systematically analyze the security and convergence guarantees of our scheme. The theoretical proofs are presented in Appendix A of the full version[1].

A. Security Analysis

With regard to the privacy property, we establish the security of our protocol as a secure multiparty computation. This is demonstrated by showing that the joint view of the server and any subset of fewer than t clients reveals no information about the inputs of other clients, except for what can be inferred from the output of the computation.

Let $\mathcal{U}$ be the set of clients, and $\mathcal{C} \subset \mathcal{U} \cup \{\mathcal{S}\}$ be the set of honest-but-curious parties, x_u be the input of client u. Suppose the simulator SIM has access to an oracle $\mathsf{IDEAL}(t, x_u)_{u \in \mathcal{U} \setminus \mathcal{C}}$ as defined below:

$$\mathsf{IDEAL}(t, x_u)_{u \in \mathcal{U} \setminus \mathcal{C}} = \begin{cases} \sum\limits_{u \in \mathcal{U} \setminus \mathcal{C}} x_u & |\mathcal{U} \setminus \mathcal{C}| > t \\ \perp & \text{otherwise} \end{cases}$$

Remember that in the traditional SecAgg scheme, the pairwise seeds used to generate masks are shared among users. For the (t, n) Shamir scheme with a shared secret s, the secret can only be reconstructed if more than t or more shares are combined. Similarly, in the cluster aggregation scheme, we require that there should not be too many type I attackers in each cluster to prevent the reconstruction of honest users' secrets. When analyzing the security of our scheme, we need to provide a precondition to ensure its security.

Theorem 1. *Let n be the number of clients, k be the number of clusters, l be the number of members in each cluster, t be the Shamir threshold, and q_1 be the number of type I attackers. Thus, given the security parameter ξ, the constraint on k can be expressed as:*

$$k < \frac{2^{-\xi}}{\sum_{i=t}^{min\{q_1, l-1\}} \frac{q_1!(n-q_1)!l!(n-l)!}{i!(q_1-i)!(l-i)!(n-q_1-l+i)!n!}} \tag{23}$$

[1]Please refer to the full version: https://eprint.iacr.org/2024/2082

ClusterGuard

- **System Setup:**
  - Inputs: security parameter λ, current federated learning training round r, local model update $\mathbf{x}_u$.
  - Outputs: secure aggregated model update $\mathbf{z}$.
  1) Initialize Public Key Infrastructure (PKI), assign each client $u \in \mathcal{U}$ a unique identity and key pairs (epk_u, esk_u) for encryption, (rpk_u, rsk_u) for VRF.
  2) Set public parameters $\{\ell, \mathfrak{p}, \mathfrak{q}, c\}$ for LKH-PRF$_{\mathbf{k}}$.
  3) Configure VRF's bilinear pairing $e : G_1 \times G_2 \to G_T$ and public hash functions.
  4) Invoke $\mathbf{VSS.Setup}(1^\lambda)$ to initialize public parameters (G, q, g) with threshold t, where $q = \mathfrak{q}$.
  5) Set the number of clusters k, cluster size l, and scaling factor Δ.

1) **Round 0 - Initialization:**
   Client u:
   a) Generate $m_u \in \{0,1\}^*$ and computes $(r_u, \pi_u) \leftarrow \mathsf{Eval}(rsk_u, m_u)$. Broadcast π_u to other clients via the server. Receive $\{\pi_v\}_{v \in \mathcal{U} \setminus u}$ and broadcast (m_u, r_u) to other clients via the server.
   b) Receive $\{m_v, r_v\}_{v \in \mathcal{U} \setminus u}$ and perform VRF verification, if verification fails, abort.
   c) Compute $R_{\text{total}} = \bigoplus_{u \in \mathcal{U}} r_u$ and determine own cluster $\mathcal{C}_\varphi$ via $\varphi_u = \text{Hash}(u || R_{\text{total}} || r) \bmod k$.
   d) Generate vector key $\mathbf{s}_u \xleftarrow{\$} \mathbb{Z}_{\mathfrak{q}}^{\ell}$ and compute $\mathbf{c} = (c||1, c||2, \cdots, c||m)$.

2) **Round 1 - Key sharing:**
   Client u:
   a) Generate $|\mathcal{C}_\varphi|$ key shares for each element s_u in $\mathbf{s}_u$: $(\{v, s_{u,v}\}_{v \in \mathcal{C}_\varphi}, \{g^{s_{u,v}}\}_{v \in \mathcal{C}_\varphi}, \{g^{a_{u,d}}\}_{d \in [0, t-1]}) \leftarrow \mathbf{VSS.Share}(s_u, t, \mathcal{C}_\varphi)$.
   b) Send $\mathsf{Enc}(epk_v, v \| s_{u,v})$ to $v \in \mathcal{C}_\varphi \setminus \{u\}$ via the server, broadcast $(\{g^{s_{u,v}}\}_{v \in \mathcal{C}_\varphi}, \{g^{a_{u,d}}\}_{d \in [0,t-1]})$ within $\mathcal{C}_\varphi$ via the server.
   c) Receive shares $\{\mathsf{Enc}(epk_u, u \| s_{v,u})\}_{v \in \mathcal{C}_\varphi}$, decrypt with esk_u to obtain $\{u, s_{v,u}\}_{v \in \mathcal{C}_\varphi}$, receive $(g^{s_{v,u}}, \{g^{a_{v,d}}\}_{d \in [0,t-1]})$ from $v \in \mathcal{C}_\varphi$, check shares received: $\mathbf{VSS.Verify}(u, g^{s_{v,u}}, \{g^{a_{v,d}}\}_{d \in [0,t-1]}) = 1$.
   d) Store all messages received, and if any of the above operations (share, check) fails, abort.

   Server:
   a) For each cluster $\mathcal{C}_\varphi \subset \mathcal{U}$, receive $(u, \{g^{s_{u,v}}\}_{v \in \mathcal{C}_\varphi})$ from at least t clients (denote with $u \in \mathcal{C}_\varphi^{(1)} \subseteq \mathcal{C}_\varphi$ and $\mathcal{U}^{(1)} \subseteq \mathcal{U}$).
   b) Store all messages received.

3) **Round 2 - Model update collection:**
   Client u:
   a) Compute the masked input vector: $\mathbf{y}_u = \Delta \cdot \text{Encode}(\mathbf{x}_u) + \text{LKH-PRF}_{\mathbf{s}_u}(\mathbf{c})$, and send $\mathbf{y}_u$ to server.

   Server:
   a) For each cluster $\mathcal{C}_\varphi^{(1)} \subset \mathcal{U}^{(1)}$, collect the masked model $\mathbf{y}_u$ from as least t clients (denote with $\mathcal{C}_\varphi^{(2)} \subseteq \mathcal{C}_\varphi^{(1)}$ and $\mathcal{U}^{(2)} \subseteq \mathcal{U}^{(1)}$).
   b) For each cluster $\mathcal{C}_\varphi^{(2)} \subset \mathcal{U}^{(2)}$, if $|\mathcal{C}_\varphi^{(2)}| \geq t$, compute $\mathbf{y}_{\mathcal{C}_\varphi^{(2)}} = \sum_{u \in \mathcal{C}_\varphi^{(2)}} \mathbf{y}_u$, and send to each client $u \in \mathcal{C}_\varphi^{(2)}$ its cluster's information, i.e., an online client list. Otherwise, remove $\mathcal{C}_\varphi^{(2)}$ from $\mathcal{U}^{(2)}$.

4) **Round 3 - Unmasking:**
   Client u:
   a) Receive from the server its new cluster's information, i.e., an online client list $\{v\}_{v \in \mathcal{C}_\varphi^{(2)}}$.
   b) Accumulate shares received from $v \in \mathcal{C}_\varphi^{(2)}$: $(u, s_u^*) \leftarrow \mathbf{VSS.Accum}(\{u, s_{v,u}\}_{v \in \mathcal{C}_\varphi^{(2)}})$, and send $(u, s_u^*, g^{s_u^*})$ to the server.

   Server:
   a) For each cluster $\mathcal{C}_\varphi^{(2)} \subset \mathcal{U}^{(2)}$, receive $(u, s_u^*, g^{s_u^*})$ from at least t clients (denote with $\mathcal{C}_\varphi^{(3)} \subseteq \mathcal{C}_\varphi^{(2)}$ and $\mathcal{U}^{(3)} \subseteq \mathcal{U}^{(2)}$).
   b) For each client $u \in \mathcal{C}_\varphi^{(3)}$, check shares: $\mathbf{VSS.VrfyAccum}(u, g^{s_u^*}, \{g^{s_{v,u}}\}_{v \in \mathcal{C}_\varphi^{(1)}}) = 1$.
   c) For each cluster $\mathcal{C}_\varphi^{(3)} \subset \mathcal{U}^{(3)}$, if $|\mathcal{C}_\varphi^{(3)}| \geq t$, recover $s_{\mathcal{C}_\varphi^{(2)}} \leftarrow \mathbf{VSS.Recon}(\{u, s_u^*\}_{u \in \mathcal{C}_\varphi^{(3)}})$ to construct the ℓ-dimensional vector key $\mathbf{s}_{\mathcal{C}_\varphi^{(2)}}$. Otherwise, remove $\mathcal{C}_\varphi^{(2)}$ from $\mathcal{U}^{(2)}$.

   Case 1: Without considering robust aggregation
   a) For each cluster $\mathcal{C}_\varphi^{(2)} \subset \mathcal{U}^{(2)}$, compute the mask $\mathbf{s} = \sum_{\mathcal{C}_\varphi^{(2)} \subset \mathcal{U}^{(2)}} \mathbf{s}_{\mathcal{C}_\varphi^{(2)}} = \sum_{u \in \mathcal{U}^{(2)}} \mathbf{s}_u$.
   b) Unmask the aggregated model update: $\hat{\mathbf{z}} = \frac{1}{n'\Delta} \left(\sum_{\mathcal{C}_\varphi^{(2)} \subset \mathcal{U}^{(2)}} \mathbf{y}_{\mathcal{C}_\varphi^{(2)}} - \text{LKH-PRF}_{\mathbf{s}}(\mathbf{c}) \right)$, where $n' = |\mathcal{U}^{(2)}|$, and $\mathbf{c} = (c \| 1, c \| 2, \cdots, c \| m)$.
   c) If any of the above operation (check, recon, LKH-PRF, unmask) fails, abort.
   d) Decode $\hat{\mathbf{z}}$ to output $\mathbf{z} = \frac{1}{n'} \sum_{u \in \mathcal{U}^{(2)}} \mathbf{x}_u$.

   Case 2: Considering robust aggregation
   a) For each cluster $\mathcal{C}_\varphi^{(2)} \subset \mathcal{U}^{(2)}$, perform mask elimination for cluster update aggregation: $\hat{\mathbf{z}}_{\mathcal{C}_\varphi^{(2)}} = \frac{1}{|\mathcal{C}_\varphi^{(2)}| \Delta} \left(\mathbf{y}_{\mathcal{C}_\varphi^{(2)}} - \text{LKH-PRF}_{\mathbf{s}_{\mathcal{C}_\varphi^{(2)}}}(\mathbf{c}) \right)$, where $\mathbf{c} = (c \| 1, c \| 2, \cdots, c \| m)$.
   b) If any of the above operation (check, recon, LKH-PRF, unmask) fails, abort.
   c) Decode $\hat{\mathbf{z}}_{\mathcal{C}_\varphi^{(2)}}$ to $\mathbf{z}_{\mathcal{C}_\varphi^{(2)}}$. Output through the robust aggregation algorithm $\mathbf{z} \leftarrow \mathbf{RobustAgg}(\{\mathbf{z}_{\mathcal{C}_\varphi^{(2)}}\})$.

Fig. 3. The description of ClusterGuard protocol for one FL round. Underlined parts are optional to guarantee public verification.

Remark. The theorem establishes the quantitative relationship between the privacy security parameter ξ and the system parameters $\{n, k, l, t\}$. The security parameter ξ controls the upper bound on the probability of privacy leakage, which is given by $2^{-\xi}$. For example: (1) with $t = \lfloor 0.8l \rfloor$, $n = kl = 100$, an adversary proportion $\delta_D = 10\%$ and $l = 10$, setting $\xi = 23$ ensures the probability is bounded above by 2^{-23}. (2) with $t = \lfloor 0.8l \rfloor$, $n = kl = 100$, an adversary proportion $\delta_D = 20\%$ and $l = 20$, setting $\xi = 33$ ensures the probability is bounded above by 2^{-33}.

Then the security of our protocol is given by the following theorem:

Theorem 2 (Local Model Privacy Under Type I Attackers)**.** *There exists a PPT simulator SIM such that for all PPT adversaries $\mathcal{A}_\mathcal{C}$, all λ, t, $\mathcal{U}$, x_u, and $\mathcal{C}$ such that $\mathcal{C} \subseteq \mathcal{U} \cup \mathcal{S}$ and $|\mathcal{C} \setminus \mathcal{S}| < t$, the output of SIM is computationally indistinguishable from the output of $\mathsf{REAL}_\mathcal{C}^{\mathcal{U},t,\lambda}$:*

$$\mathsf{REAL}_\mathcal{C}^{\mathcal{U},t,\lambda}(\mathcal{A}_\mathcal{C}, x_{\mathcal{U}\setminus\mathcal{C}}) \approx_c SIM_\mathcal{C}^{\mathcal{U},t,\lambda,\mathsf{IDEAL}(t,x_u)_{u\in\mathcal{U}\setminus\mathcal{C}}}(\mathcal{A}_\mathcal{C})$$

B. Convergence Analysis

The goal of federated learning is to collaboratively train a global model across multiple clients such that the expected loss function over all clients is minimized. The formal definition is as follows:

$$\theta^* = \min_\theta \mathbb{E}_{\xi\sim\mathcal{D}}\left[\mathcal{L}(\theta, \xi)\right] \tag{24}$$

where θ^* denotes the optimal global model, λ represents the training data sampled from $\mathcal{D} = \cup_{u\in\mathcal{U}}\mathcal{D}_u$, $\mathcal{L}(\theta, \xi)$ is the loss function of the model θ under the data ξ.

We use $F(\theta)$ to represent $\mathbb{E}_{\xi\sim\mathcal{D}}\left[\mathcal{L}(\theta, \xi)\right]$. Before providing the convergence proof, we follow the common assumptions for distributed optimization convergence analysis given in [46].

Assumption 1. *$L-Lipschitz$ continuity of the gradient: the gradient of $F(\theta)$ is Lipschitz continuous with constant $L > 0$, for all $\theta, \theta' \in \mathbb{R}^d$,*

$$F(\theta') \leq F(\theta) + \nabla F(\theta)^\top(\theta' - \theta) + \frac{L}{2}\|\theta' - \theta\|^2 \tag{25}$$

Assumption 2. *Variance between local and global model updates: There exists a constant σ_g^2 such that for all clients u_i, the variance between the client model updates $\nabla F_i(\theta)$ and the global model update $\nabla F(\theta)$ is bounded:*

$$\mathbb{E}\left[\|\nabla F_i(\theta) - \nabla F(\theta)\|^2\right] \leq \sigma_g^2 \tag{26}$$

The stochastic gradient of each worker (client) has a bounded variance uniformly, satisfying:

$$\mathbb{E}_{\xi_i\sim\mathcal{D}_i}\left[\|\nabla F_i(\theta; \xi_i) - \nabla F_i(\theta)\|^2\right] \leq \sigma_c^2 \tag{27}$$

where $F_i(\theta; \xi_i)$ is the stochastic gradient computed by worker u_i based on the sample ξ_i, and $\nabla F_i(\theta)$ is the expesected gradient for worker u_i.

Our robustness algorithm uses model updates trained on trusted root dataset or datasets from honest clients to guide trust. It leverages the similarity between the aggregation of each cluster and this trusted model update to filter out the clusters containing malicious clients. Referencing the work [47] [46], any robust gradient aggregation rule inevitably results in an error in the average honest gradient. Therefore, we propose the following assumption:

Assumption 3. *For the problem with $(1-\beta)k$ benign clusters (without malicious clients) (denoted by $\mathcal{G}_g$) and βk Byzantine clusters (with malicious clients), suppose at most δk Byzantine clusters can bypass the robustness algorithm at each iteration. Assume the existence of positive constants c and b such that the model update output $\boldsymbol{RobustAgg}(\{\boldsymbol{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}})$ from the robustness algorithm satisfies the following conditions:*

Bounded Bias:

$$\mathbb{E}\left[\|\boldsymbol{RobustAgg}(\{\boldsymbol{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}}) - \bar{\boldsymbol{w}}\|\right]^2 \leq c\delta \sup_{\mathcal{C}_i,\mathcal{C}_j\in\mathcal{G}_g} \mathbb{E}\left[\|\boldsymbol{w}_r^{(\mathcal{C}_i)} - \boldsymbol{w}_r^{(\mathcal{C}_j)}\|^2\right] \tag{28}$$

where $\bar{\boldsymbol{w}}_r = \frac{1}{|\mathcal{G}_g|}\sum_{\mathcal{C}_i\in\mathcal{G}_g}\boldsymbol{w}_r^{(\mathcal{C}_i)}$.

Bounded Variance:

$$Var\left[\|\boldsymbol{RobustAgg}(\{\boldsymbol{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}})\|\right] \leq b^2 \tag{29}$$

Here, δ is the fraction of Byzantine clusters that can bypass the robustness algorithm, and $0 \leq \delta < \beta < 1$.

Theorem 3. *Consider the function F that satisfies assumptions 1 and 2. Assume there is a robust aggregation scheme that satisfies assumption 3. Let the number of type II attackers be q_2, and $F^* = min_\theta\nabla(\theta)$. There exists $\eta < \frac{1}{L}(1 - \frac{\sqrt{\delta}}{4} - \frac{q_2}{2n})$ for which, after R rounds, the following condition holds:*

$$\frac{1}{R}\sum_{r=0}^{R-1}\mathbb{E}\|\nabla F(\theta_r)\|^2 \leq \frac{(F(\theta_0) - F^*)}{M\eta R} + \frac{L\eta}{M}\Delta_3 + \frac{1}{M}\Delta_4 \tag{30}$$

where $M = 1 - \frac{\sqrt{\delta}}{4} - \frac{q_2}{2n} - L\eta, \Delta_4 = \frac{8c\sqrt{\delta}}{l}(\sigma_c^2 + \sigma_g^2) + \frac{nq_2\sigma_g^2}{2(n-q_2)^2}, \Delta_3 = 2\Delta_1 + 4\Delta_2 + 12\frac{n-q_2-(1-\beta)kl}{(n-q_2-1)(1-\beta)kl}(\sigma_c^2 + \sigma_g^2 + \Delta_2), \Delta_1 = \frac{8c\delta}{l}(\sigma_c^2 + \sigma_g^2) + b^2, \Delta_2 = \frac{q_2^2\sigma_g^2}{(n-q_2)^2} + \frac{2\sigma_c^2}{n-q_2}$.

Remark. The theorem provides an insightful analysis of federated learning convergence under adversarial settings, with and without attacks. In the ideal case where $\delta = \beta = 0$ and $q_2 = 0$, indicating no adversarial behavior, the convergence bound simplifies significantly. In this scenario, the parameter $M = 1 - L\eta$ depends only on the smoothness constant L and the learning rate η, with the error terms $\Delta_3 = 2b^2 + \frac{8\delta_c^2}{n}, \Delta_4 = 0$. At this point, setting a sufficiently small learning rate can ensure convergence without extra errors. Specifically, if we set $\eta = \mathcal{O}(\frac{1}{\sqrt{R}})$, and in this case, we can obtain $\frac{1}{R}\sum_{r=0}^{R-1}\mathbb{E}\|\nabla F(\theta_r)\|^2 \leq \mathcal{O}(\frac{F(\theta_0)-F^*}{\sqrt{R}-L} + \frac{L}{\sqrt{R}-L}\Delta_3)$. At this point, the ClusterGuard scheme will converge with a rate of $\mathcal{O}(\frac{1}{\sqrt{R}})$, and the extra error terms will also be eliminated.

In the general case, where $\delta > 0$, $\beta > 0$, and $q_2 > 0$, the convergence behavior is influenced by adversarial robustness and the number of attackers. We still take $\eta = \mathcal{O}(\frac{1}{\sqrt{R}})$, and we can obtain: $\frac{1}{R}\sum_{r=0}^{R-1}\mathbb{E}\|\nabla F(\theta_r)\|^2 \leq \mathcal{O}(\frac{(F(\theta_0)-F^*)}{\left(\sqrt{R}(1-\frac{\sqrt{\delta}}{4}-\frac{q_2}{2n})-L\right)} + \frac{L\Delta_3}{\left(\sqrt{R}(1-\frac{\sqrt{\delta}}{4}-\frac{q_2}{2n})-L\right)} + \frac{\Delta_4}{1-\frac{\sqrt{\delta}}{4}-\frac{q_2}{2n}-\frac{L}{\sqrt{R}}})$. The overall convergence rate can be observed as $\mathcal{O}(\frac{1}{\sqrt{R}})$, with an additional error

term $\frac{\Delta_4}{1-\frac{\sqrt{\delta}}{4}-\frac{q_2}{2n}}$. When malicious clients (i.e., q_2 increases), the robustness of the aggregation algorithm is insufficient (i.e., δ increases), or the heterogeneity among client updates increases (i.e., σ_g^2, σ_c^2 increases), the extra error term will grow.

VI. Discussion

A. Decentralized Mode Extension

Our cluster-based federated learning scheme can be extended to decentralized federated learning scenarios, possibly with the help of blockchains. Accordingly, we need to revise the protocol as follows:

Cluster Formation. Each client u holds a key pair (pk_u, sk_u). The client generates a random byte string m_u and computes a verifiable random value and proof (r_u, π_u) using its key pair. It then constructs a message tuple (m_u, r_u, π_u, r) incorporating the current federated learning round r, and broadcasts it to other clients. Upon receiving the broadcast, other clients perform verification. If the verification is successful, each client determines the cluster configuration $\{\mathcal{C}_\varphi\}_{\varphi\in[1,k]}$ and cluster heads $\{u_{\mathcal{C}_\varphi}\}_{\varphi\in[1,k]}$ for the current round as follows: (1) compute the total randomness as $R_{\text{total}} = \bigoplus_{i=1}^{n} r_i$, where $\bigoplus$ denotes bitwise XOR. (2) assign the cluster index using a hash function: $\varphi_u = \text{Hash}(u\|R_{\text{total}}\|r) \mod k$, where φ_u indicates the cluster to which client u_i belongs, denoted $\mathcal{C}_\varphi$. (3) select the cluster head using a secure pseudorandom function (PRF): $u_{\mathcal{C}_\varphi} = \text{PRF}(R_{\text{total}}, r\|\varphi) \mod l$, where φ is the cluster index and $u_{\mathcal{C}_\varphi}$ is the designated cluster head of $\mathcal{C}_\varphi$.

Secure Aggregation. Each cluster head $u_{\mathcal{C}_\varphi}$ securely aggregates the updates $\mathbf{z}_{\mathcal{C}_\varphi}$ within its cluster $\mathcal{C}_\varphi$ (note that the cluster head also participates in federated training). After aggregation, all cluster heads exchange their aggregated updates $\{\mathbf{z}_{\mathcal{C}_\varphi}\}_{\varphi\in[1,k]}$ and distribute the unified aggregated updates to all clients.

Robust Aggregation. In the robust aggregation phase, each client filters out cluster updates identified as malicious using a robustness algorithm, and performs robust aggregation over the received cluster updates $\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}}$, which are reconstructed from the aggregated vectors $\{\mathbf{z}_{\mathcal{C}_\varphi}\}_{\mathcal{C}_\varphi\subset\mathcal{U}}$. Specifically, each client filters updates based on its local dataset $\mathcal{D}_u$:

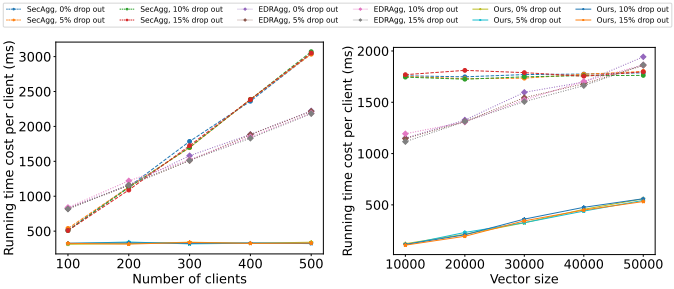
$$\mathbf{w}_r^{(u)} \leftarrow \textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}}, \mathcal{D}_u) \tag{31}$$

Here, **RobustAgg**$(\cdot)$ denotes the proposed adaptive dual-threshold filtering algorithm, designed to identify and eliminate abnormal updates. The detailed algorithm is provided in Algorithm 1. Unlike the centralized setting, in the decentralized scenario the algorithm relies on each client's own validated local dataset $\mathcal{D}_u$ for trust evaluation, instead of a server-maintained trusted root dataset $\mathcal{D}_0$.

Detailed evaluation results for these scenarios are illustrated in Appendix B[2].

VII. Evaluation

We analyze the robust aggregation of ClusterGuard with n clients and a single server, where each client holds a data vector of size m and a secret key of size ℓ. For clients, the computation cost is $O(\ell l^2 + m\ell + n)$, which includes generating a secret vector, creating l secret shares, computing the model update mask and verifying all VRFs. The communication cost is $O(\ell l + m + n)$, covering the exchange of secret shares, verification, masked updates, and VRFs. For the server, the computation cost is $O(nl + \frac{nm\ell}{l})$, dominated by reconstructing cluster secret keys and calculating masks, which can reduce to $O(nt + m\ell + \frac{n\ell}{l})$ without robustness detection. If some clients drop out, the cost further decreases. The communication cost is $O(mn + \ell n)$ for receiving masked updates, secret shares, and sending cluster information.



(a) Running time cost, as the number of clients increases.

(b) Running time cost, as the size of model update vector increases.

Fig. 4. Client protocol performance comparison. The data vector size remains fixed at 30K in (a), while the number of clients is fixed at 300 in (b).

A. Protocol Performance

Our experimental setup consists of two parts. First, we evaluate the performance of the secure aggregation protocol without considering users' local training processes. Second, we assess federated learning accuracy and robustness using the secure aggregation protocol, involving complete federated learning training.

1) Experiment Setup: The experiments are conducted on a Windows desktop with AMD Ryzen 9 5900HX (3.3 GHz), with 32 GB of RAM and NVIDIA GeForce RTX 3080 Laptop GPU. The main aggregation protocol was implemented in Python. The lattice-based key-homomorphic pseudorandom function (LKH-PRF) was implemented using the GoFE library[3] and was accelerated via goroutines. The precision of model updates was set to 16 bits by default (for encoding and decoding operations, we set $prec = 16$, $\mathbf{b} = \mathbf{8}$). In the LKH-PRF scheme, the parameters of $\text{LWR}_{\ell,\mathfrak{q},\mathfrak{p}}$ were configured as $\ell = 512$, $\mathfrak{p} = 2^{35}$, and $\mathfrak{q}$ as a 70-bit prime close to 2^{70}. According to estimations based on the LWE estimator [9], [48], the hardness[4] of this setting exceeds 2^{128}. For secure channel encryption in the public key infrastructure, we use AES-GCM with 128-bit keys. The cluster formation based on verifiable randomness was implemented using the bn256 library[5]. For each cluster, the default parameters were set as $l = 10$ and $t = \lfloor 0.8l \rfloor$.

[2]Please refer to the full version: https://eprint.iacr.org/2024/2082

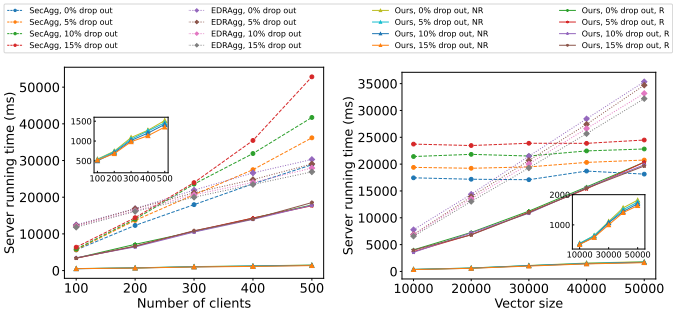
[3]https://github.com/fentec-project/gofe

[4]based on the hybrid lattice-reduction and meet-in-the-middle attack

[5]https://github.com/cloudflare/bn256

2) Experimental Performance: Figure 4 compares client-side time overhead under different dropout rates. Compared to SecAgg (Bonawitz et al. [2]) and EDRAgg (Liu et al. [8]), our protocol reduces client-side runtime by over $2\times$. As the number of clients increases (Figure 4(a)), the overhead per client remains nearly constant, since each client only computes its model update mask and shares keys with its cluster, independent of client count. SecAgg requires key exchange and mask sharing with other clients, while EDRAgg involves additional mask calculations and secret sharing.

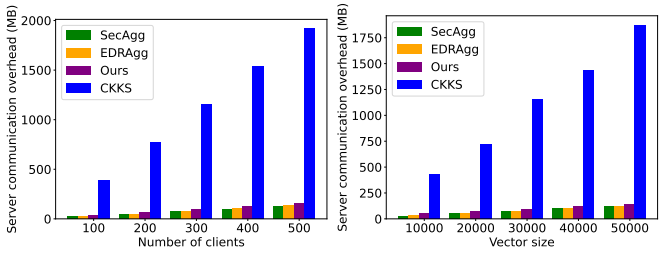
In Figure 4(b), SecAgg's runtime remains stable because it employs pseudorandom number calculation under AES CTR mode (utilizing one-time filling), which does not significantly vary with vector size as the number of clients is fixed. In contrast, while the proposed method and EDRAgg show a similar increase in client-side overhead with more model parameters, our protocol still outperforms both SecAgg and EDRAgg. This increase is attributed to the fact that calculating a single mask entails m ℓ-dimensional vector inner product operations, with computation time directly proportional to vector size, similar to EDRAgg. Figure 5(a) shows that as the



(a) Server time cost vs number of clients, the vector size is fixed at 30K.

(b) Server time cost vs vector size, the number of clients is fixed at 300.

Fig. 5. Server protocol performance comparison. R denotes the server with robust aggregation, while NR denotes non-robust aggregation.

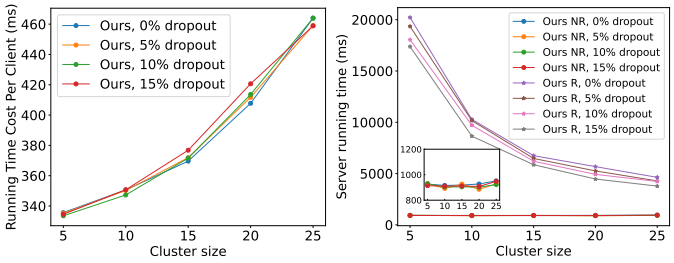


(a) Server communication overhead, as the number of clients increases.

(b) Server communication overhead, as the vector size increases.

Fig. 6. Server communication overhead comparison. The data vector size remains fixed at 30K in (a), while the number of clients is fixed at 300 in (b).

number of clients increases, our protocol significantly outperforms SecAgg and EDRAgg, both when robust aggregation is considered and when it is not. Without robust aggregation, the server only needs to compute a single mask, resulting in lower overhead. With robust aggregation, the server incurs additional overhead to compute the total mask for all clusters. Figure 5(b) illustrates the relationship between server runtime and model update size. Without robust aggregation, our protocol requires only a single mask recovery, significantly reducing runtime compared to SecAgg and EDRAgg. With robust aggregation, the runtime increases but remains acceptable compared to the other protocols. Overall, as the dropout rate increases, SecAgg's server time increases significantly, while our protocol remains stable. This is because SecAgg requires additional pairwise mask recovery for dropped clients, while our protocol only needs to wait for communication from t clients. For EDRAgg, although server time decreases with increased dropout rate, it still incurs additional model aggregation time (multiplicative rather than additive) and the time to solve aggregation results (using Pollard's lambda method [49], which requires roughly square root of time in the plaintext space). This extra time overhead becomes significant, especially for large model sizes. Figure 6 shows the



(a) Running time per client, as the cluster size increases.

(b) Total server running time, as the cluster size increases.

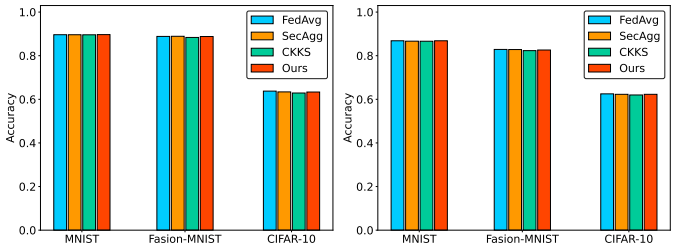
Fig. 7. The influence of cluster size on the performance of the protocol. The data vector size remains fixed at 30K and the number of clients is fixed at 300. R denotes the server with robust aggregation, while NR denotes non-robust aggregation.

comparison of server communication overhead, assuming no client dropout for a fair comparison. Although our protocol uses vector keys as the secret sharing communication unit, resulting in slightly higher total data transmission than SecAgg and EDRAgg, its communication overhead is still significantly lower than that of aggregation schemes based on CKKS homomorphic encryption. Lastly, Figure 7 shows the impact of cluster size on runtime. Reducing the cluster size decreases client runtimes, while increasing it reduces server runtime with robustness detection. This is because larger clusters increase the client computation for secret sharing, while reducing the server's workload for mask recovery due to fewer clusters. Thus, properly adjusting the cluster size enables a balanced computational load between clients and the server, improving overall system efficiency.

Table III presents the experimental results under privacy attacks (DLG [1] and iDLG [50]). We conduct evaluations on the MNIST and CIFAR datasets, with a batch size of 1 (note that increasing the batch size makes the attack more difficult [1], so this setting represents the worst-case scenario). Without protection (FedAvg), recognizable images emerged after 250 epochs, showing vulnerability to attacks. Then, we evaluate our scheme. For different cluster sizes, the reconstruction results are similar. We select a representative case with a cluster size of 4. Due to the application of the secure aggregation protocol, even after 500 iterations, the reconstructed images

still contain significant noise and are unrecognizable. These results confirm ClusterGuard effectively resists privacy attacks.

B. Accuracy and Robustness Performance

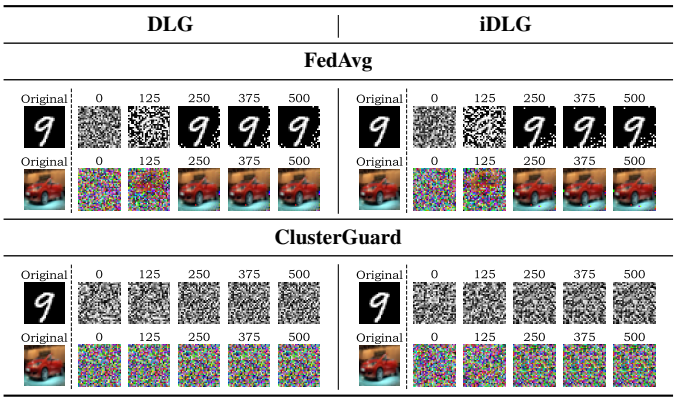


(a) IID. (b) Non-IID.

Fig. 8. Accuracy comparison under the different datasets with two different distributions.

1) Datasets and model configurations: We implement a complete federated learning process using PyTorch, based on three neural networks with varying architectures and sizes. We evaluate model accuracy and resistance to poisoning attacks using our ClusterGuard protocol on three datasets: MNIST, Fashion-MNIST, and CIFAR. For MNIST, we utilize a Logistic Regression model with 7,850 parameters. For Fashion-MNIST, we employ the standard LeNet [51] architecture, which contains 61,706 parameters. Finally, for CIFAR, we use a large-scale CNN with 160,856 parameters.

TABLE III
RECONSTRUCTION RESULTS UNDER PRIVACY ATTACKS.



2) Federated Learning Settings: We consider two scenarios: IID and non-IID data distribution. For IID data, the dataset is evenly distributed among clients, while for non-IID data, we use a Dirichlet distribution with a hyperparameter $\alpha = 0.5$. In our experiments, we adopt a default non-IID setting. We involve $[100, 100, 60]$ clients for MNIST, Fashion-MNIST and CIFAR-10. Each client's local learning rate is $[0.01, 0.01, 0.05]$, with 5 local iterations and a batch size of 32. By default, we set $l = t = 4$.

3) Poisoning Attacks and Defense: In our algorithm experiment, we uniformly extract 500 samples as the root data set $\mathcal{D}_0$, like FLtrust [30]. For our robust algorithm, we set parameters ρ, bl_1, bl_2 to $1.5, 0.05, 2$ respectively. For the multi-round detection mechanism, the Local Outlier Factor algorithm is implemented using the scikit-learn library[6], with the default parameter $K = \frac{n}{2}$. We consider the following common poisoning attacks:

Label flipping (LF) Attack [24]: Specifically, the poisoning attackers are trained using incorrect labels, i.e., any label from $\{0, ..., 9\}$ is transformed into $9-$label.

Noise Attack [26]: The attackers perturb their gradients by adding Gaussian noise with mean 0 and standard deviation σ, i.e., $\mathbf{w}_m = \mathbf{w} + \mathcal{N}(\mu, \sigma^2)$. We take $\mu = 0, \sigma = 0.5$.

Sign flipping (SF) Attack [52]: Specifically, during the local update step, the poisoning attackers reverse the sign of the model updates, i.e., $\mathbf{w}_m = -\delta\mathbf{w}$. We take $\delta = 2$.

Scale Attack [30]: We consider an attacker uploading a scaled version of the benign model update. Typically, the scaling factor γ satisfies $\gamma \gg 1$. In our experiment, we set $\gamma = 20$.

IPM Attack [53]: The attackers craft malicious updates to induce a negative inner product between the true mean of benign client updates and the output of the robust aggregation rule, thereby disrupting the convergence of gradient descent. Assuming the attackers know the mean of the updates from the benign client set $\mathcal{U}_b$ and collude with each other, the attack is parameterized by $\epsilon = 0.8$ in our experiments. The IPM attack is constructed as follows:

$$\mathbf{w}_m = -\frac{\epsilon}{|\mathcal{U}_b|} \sum_{u_i \in \mathcal{U}_b} \mathbf{w}^{(u_i)} \tag{32}$$

Min-Max Attack [54]: The poisoning attackers send perturbed versions of the benign aggregated update. Here, ∇^p is the perturbation vector, γ is a scaling factor, and f_{avg} is the average aggregation function. The construction is as follows:

$$\mathbf{w}_m = f_{avg}(\{\mathbf{w}^{(u_i)}\}_{u_i \in \mathcal{U}}) + \gamma \nabla^p \tag{33}$$

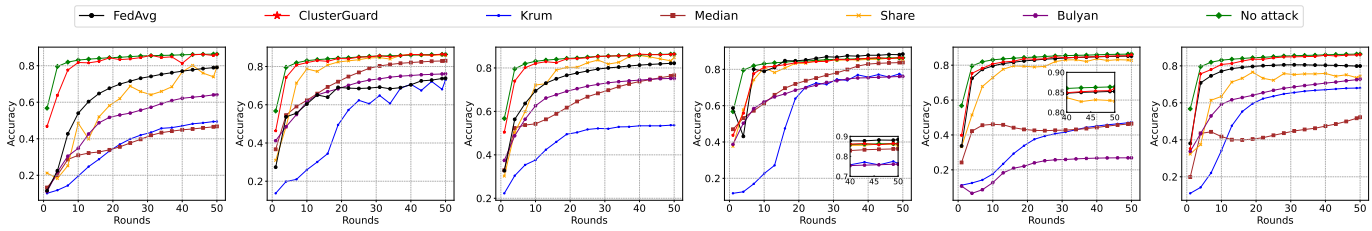
$$\underset{\gamma}{\text{argmax}} \max_{u_i \in \mathcal{U}} \|\mathbf{w}_m - \mathbf{w}^{(u_i)}\|_2 \leq \max_{u_i, u_j \in \mathcal{U}} \|\mathbf{w}^{(u_i)} - \mathbf{w}^{(u_j)}\|_2 \tag{34}$$

By default, $\nabla^p = -sign(f_{avg}(\{\mathbf{w}^{(u_i)}\}_{u_i \in \mathcal{U}}))$, where sign$(\cdot)$ denotes the sign function. To maximize the attack effectiveness, the attackers need to collude with each other to ensure that their malicious updates remain consistent.

4) Experimental Performance: We first discuss the comparison of the model accuracy of our protocol with others. Since the error of LKH-PRF has been processed, the influence of the model accuracy only comes from the encoding and decoding of the model updates. As shown in Figure 8, our protocol has the same model accuracy as plaintext aggregation (FedAvg [14]), and is also consistent with other secure aggregation protocols.

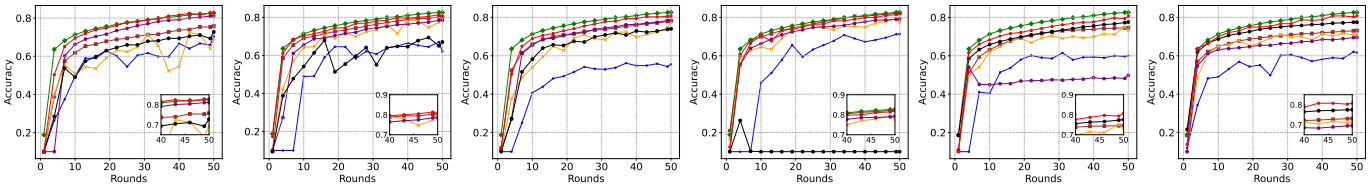
Figures 9, 10, and 11 illustrate the comparison of different defense algorithms against various attacks when the proportion of malicious clients is 20%. In our experiments, we default to using a non-IID setting for defense evaluation. The results show that our proposed aggregation method consistently achieves superior convergence accuracy compared to other robust defense methods across a comprehensive set of poisoning attacks, including LF, Noise, SF, Scale, IPM, and Min-Max.

[6]https://scikit-learn.org/stable/



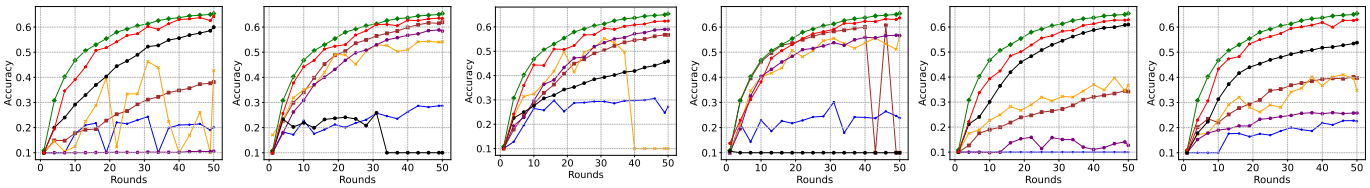
(a) LF Attack. (b) Noise Attack. (c) SF Attack. (d) Scale Attack. (e) IPM Attack. (f) Min-Max Attack.

Fig. 9. Accuracy comparison. Assuming 20% of the clients are malicious. The experimental setup involves the MNIST dataset and LR model.



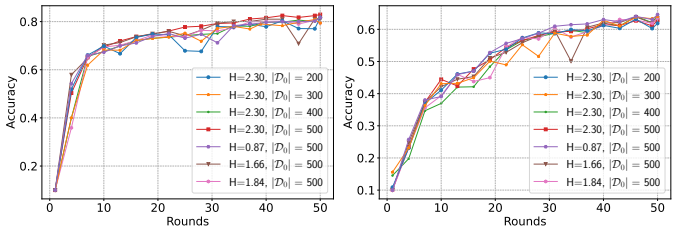
(a) LF Attack. (b) Noise Attack. (c) SF Attack. (d) Scale Attack. (e) IPM Attack. (f) Min-max Attack.

Fig. 10. Accuracy comparison. Assuming 20% of the clients are malicious. The experimental setup involves the Fashion-MNIST dataset and LeNet model.



(a) LF Attack. (b) Noise Attack. (c) SF Attack. (d) Scale Attack. (e) IPM Attack. (f) Min-max Attack.

Fig. 11. Accuracy comparison. Assuming 20% of the clients are malicious. The experimental setup involves the CIFAR dataset and CNN model.



(a) Fasion-MNIST. (b) CIFAR-10.

Fig. 12. The impact of root dataset with different distributions on the accuracy of global model.

TABLE IV
THE IMPACT OF ALGORITHM-RELATED PARAMETERS ON MODEL ACCURACY (%).

Dataset	Scheme	Cluster size				
		3	4	5	6	10
Fashion-MNIST	Share	66.09	71.55	70.92	67.26	56.06
	ClusterGuard	80.25	82.39	82.20	82.30	78.94
CIFAR-10	Share	37.07	51.49	44.11	44.42	28.35
	ClusterGuard	65.39	64.11	64.27	63.42	62.09
Dataset	**Scheme**	**Threshold parameter ρ**				
		1	1.5	2	2.5	3
Fashion-MNIST	ClusterGuard	82.06	82.19	82.09	77.98	77.26
CIFAR-10	ClusterGuard	63.98	64.11	63.77	60.54	60.38

We further discuss the relationship between parameter selection and robustness of our proposed scheme in Table IV, with a default setting of 20% malicious clients performing label-flipping attacks. It shows that our ClusterGuard scheme consistently outperforms the baseline. However, when the number of attackers is significantly lower than the number of clusters, reducing the cluster size does not provide substantial benefits, as the server can already accurately locate the clusters containing adversaries. Conversely, increasing the cluster size, which causes the number of attackers to exceed the number of clusters, results in a loss of model performance. At the same time, the threshold parameter needs to be selected appropriately. When the threshold parameter is large enough, the algorithm degenerates into the FedAvg [14]. Figure 14 discusses the impact of the root dataset on the effectiveness of our robust defense approach. We conducted experiments on root datasets of varying scales and distributions, using label-flipping attacks as the evaluation scenario. We define the uniformity of the root dataset using distribution entropy, specifically, $H(\mathcal{D}) = -\sum_{i=1}^{n'} p_i \log p_i$ where p_i is the probability of the i-th class, n' is the total number of classes. The results demonstrate that differences in data distribution within the root dataset $\mathcal{D}_0$ do not affect the correctness of the algorithm. The influence of data distribution differences on the results can be relatively ignored, which further highlights the robustness and practicality of our approach. The server does

not need to obtain $\mathcal{D}_0$ with a large number of samples, and the data distribution of $\mathcal{D}_0$ does not need to be fully aligned with that of the clients. This reduces the difficulty of acquiring the root dataset for the server.

We evaluate our proposed multi-round detection mechanism under different poisoning attacks. Table V reports precision, recall, F1-score, and accuracy at round $r_t = 35$. On MNIST and Fashion-MNIST, it achieves perfect recall (1.000) under all attacks except IPM, indicating no missed detections. For noise attacks, all metrics reach 1.000, showing zero false positives. Under sign flipping and scale attacks, F1-scores remain above 0.974, confirming robustness against stronger adversarial behaviors. On Fashion-MNIST, label flipping and Min-Max attacks introduce slight performance drops, but F1-scores still reach 0.947 and 0.930, respectively, reflecting detection stability. On CIFAR-10, the method achieves perfect detection (F1-score = 1.000) under all attacks except label flipping, where it still maintains a high F1-score of 0.917. The method consistently demonstrates strong detection performance across all datasets and attacks.

TABLE V
MULTI-ROUND DETECTION PERFORMANCE ON DIFFERENT ATTACKS

Dataset	Attack	Evaluation Metrics			
		Precision	Recall	F1-score	Accuracy
MNIST	LF	1.000	1.000	1.000	1.000
	Noise	1.000	1.000	1.000	1.000
	SF	0.950	1.000	0.974	0.990
	Scale	0.950	1.000	0.974	0.990
	IPM	0.947	0.900	0.923	0.970
	Min-Max	1.000	1.000	1.000	1.000
Fashion-MNIST	LF	0.900	1.000	0.947	0.980
	Noise	1.000	1.000	1.000	1.000
	SF	0.950	1.000	0.974	0.990
	Scale	1.000	1.000	1.000	1.000
	IPM	1.000	1.000	1.000	1.000
	Min-Max	0.869	1.000	0.930	0.970
CIFAR-10	LF	0.917	0.917	0.917	0.980
	Noise	1.000	1.000	1.000	1.000
	SF	1.000	1.000	1.000	1.000
	Scale	1.000	1.000	1.000	1.000
	IPM	1.000	1.000	1.000	1.000
	Min-Max	1.000	1.000	1.000	1.000

VIII. CONCLUSION

In summary, this paper focused on the critical challenges of secure aggregation in federated learning. We highlighted the limitations of existing protocols like SecAgg, particularly in terms of computational efficiency and vulnerability to poisoning attacks. To address these concerns, we have introduced ClusterGaurd, an innovative secure aggregation scheme endowed with Byzantine robustness. ClusterGaurd employs Verifiable Random Functions for transparent clustering and leverages lattice-based key-homomorphic pseudorandom functions to facilitate efficient and secure intra-cluster aggregation. Additionally, we have devised a dual-filtering robustness algorithm to manage inter-cluster aggregation. Our extensive experiments validate the efficiency and robustness of ClusterGaurd.

REFERENCES

[1] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Advances in neural information processing systems*, vol. 32, 2019.

[2] K. Bonawitz, V. Ivanov, B. Kreuter *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[3] J. H. Bell, K. A. Bonawitz, A. Gascón *et al.*, "Secure single-server aggregation with (poly) logarithmic overhead," in *Proc. of the 2020 ACM CCS*, 2020, pp. 1253–1269.

[4] S. Kadhe, N. Rajaraman, O. O. Koyluoglu *et al.*, "Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning," *arXiv preprint arXiv:2009.11248*, 2020.

[5] J. So, B. Güler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE Journal on Selected Areas in Information Theory*, pp. 479–489, 2021.

[6] Y. Ma, J. Woods, S. Angel *et al.*, "Flamingo: Multi-round single-server secure aggregation with applications to private federated learning," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023.

[7] J. Bell, A. Gascón, T. Lepoint *et al.*, "{ACORN}: Input validation for secure aggregation," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4805–4822.

[8] Z. Liu, J. Guo, K.-Y. Lam *et al.*, "Efficient dropout-resilient aggregation for privacy-preserving machine learning," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 1839–1854, 2022.

[9] Z. Liu, S. Chen, J. Ye *et al.*, "SASH: efficient secure aggregation based on shprg for federated learning," in *Uncertainty in Artificial Intelligence*. PMLR, 2022, pp. 1243–1252.

[10] H. Li, H. Lin, A. Polychroniadou *et al.*, "LERNA: Secure single-server aggregation via key-homomorphic masking," Cryptology ePrint Archive, Paper 2023/1936, 2023.

[11] H. Lycklama, L. Burkhalter, A. Viand *et al.*, "Rofl: Robustness of secure federated learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*. IEEE, 2023, pp. 453–476.

[12] Z. Zhang, J. Li, S. Yu, and C. Makaya, "SAFELearning: Secure aggregation in federated learning with backdoor detectability," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 3289–3304, 2023.

[13] T. Stevens, C. Skalka, C. Vincent *et al.*, "Efficient differentially private secure aggregation for federated learning via hardness of learning with errors," in *31st USENIX Security Symposium*, 2022, pp. 1379–1395.

[14] B. McMahan, E. Moore, D. Ramage *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[15] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *28th Annual Symposium on Foundations of Computer Science*. IEEE, 1987, pp. 427–438.

[16] J. Ernst and A. Koch, "Private stream aggregation with labels in the standard model," *Proc. Priv. Enhancing Technol.*, no. 4, pp. 117–138, 2021.

[17] D. Boneh, K. Lewi, H. Montgomery *et al.*, "Key homomorphic prfs and their applications," in *Annual Cryptology Conference*. Springer, 2013, pp. 410–428.

[18] Y. Shi, T. Luo, J. Liang *et al.*, "Obfuscating verifiable random functions for proof-of-stake blockchains," *IEEE Trans. Dependable Secur. Comput. (TDSC)*, 2023.

[19] M. M. Breunig, H.-P. Kriegel, R. T. Ng *et al.*, "LOF: identifying density-based local outliers," in *Proc. of the 2000 ACM SIGMOD*, 2000, pp. 93–104.

[20] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 69, no. 6, p. 066138, 2004.

[21] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 19–38.

[22] P. Xu, M. Hu, T. Chen *et al.*, "Laf: Lattice-based and communication-efficient federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 2483–2496, 2022.

[23] B. Choi, J.-y. Sohn, D.-J. Han *et al.*, "Communication-computation efficient secure aggregation for federated learning," *arXiv preprint arXiv:2012.05433*, 2020.

[24] V. Tolpegin, S. Truex, M. E. Gursoy *et al.*, "Data poisoning attacks against federated learning systems," in *Proc. 25th Eur. Symp. Res. Comput. Secur. (ESORICS)*, 2020, pp. 480–501.

[25] J. Hou, F. Wang, C. Wei *et al.*, "Credibility assessment based byzantine-resilient decentralized learning," *IEEE Trans. Dependable Secur. Comput. (TDSC)*, 2022.

[26] M. Fang, X. Cao, J. Jia *et al.*, "Local model poisoning attacks to Byzantine-Robust federated learning," in *29th USENIX Security Symposium*. USENIX Association, Aug. 2020, pp. 1605–1622.

[27] P. Blanchard, E. M. El Mhamdi, R. Guerraoui *et al.*, "Machine learning with adversaries: Byzantine tolerant gradient descent," *Advances in neural information processing systems*, vol. 30, 2017.

[28] D. Yin, Y. Chen, R. Kannan *et al.*, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. Int. Conf. Mach. Learn. (ICML)*. PMLR, 2018, pp. 5650–5659.

[29] R. Guerraoui, S. Rouault *et al.*, "The hidden vulnerability of distributed learning in byzantium," in *Proc. Int. Conf. Mach. Learn. (ICML)*. PMLR, 2018, pp. 3521–3530.

[30] X. Cao, M. Fang, J. Liu *et al.*, "FLTrust: Byzantine-robust federated learning via trust bootstrapping," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*. The Internet Society, 2021.

[31] D. N. Yaldiz, T. Zhang, and S. Avestimehr, "Secure federated learning against model poisoning attacks via client filtering," *arXiv preprint arXiv:2304.00160*, 2023.

[32] X. Mu, K. Cheng, Y. Shen *et al.*, "FedDMC: Efficient and robust federated learning via detecting malicious clients," *IEEE Trans. Dependable Secur. Comput. (TDSC)*, vol. 21, no. 6, pp. 5259–5274, 2024.

[33] Y. Wang, D.-H. Zhai, and Y. Xia, "RoPE: Defending against backdoor attacks in federated learning systems," *Knowl. Based Syst.*, vol. 293, p. 111660, 2024.

[34] M. Hao, H. Li, G. Xu *et al.*, "Efficient, private and robust federated learning," in *ACSAC '21: Annual Computer Security Applications Conference*. ACM, 2021, pp. 45–60.

[35] X. Liu, H. Li, G. Xu *et al.*, "Privacy-enhanced federated learning against poisoning adversaries," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4574–4588, 2021.

[36] Y. Dong, X. Chen, K. Li *et al.*, "FLOD: oblivious defender for private byzantine-robust federated learning with dishonest-majority," in *Proc. 26th Eur. Symp. Res. Comput. Secur. (ESORICS)*, 2021, pp. 497–518.

[37] Y. Miao, Z. Liu, H. Li *et al.*, "Privacy-preserving byzantine-robust federated learning via blockchain systems," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 2848–2861, 2022.

[38] Z. Ma, J. Ma, Y. Miao *et al.*, "ShieldFL: Mitigating model poisoning attacks in privacy-preserving federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 1639–1654, 2022.

[39] Y. Miao, X. Yan, X. Li *et al.*, "RFed: Robustness-enhanced privacy-preserving federated learning against poisoning attack," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 5814–5827, 2024.

[40] X. Hao, C. Lin, W. Dong *et al.*, "Robust and secure federated learning against hybrid attacks: A generic architecture," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 1576–1588, 2024.

[41] O. Aramoon, P.-Y. Chen, G. Qu *et al.*, "Meta federated learning," *arXiv preprint arXiv:2102.05561*, 2021.

[42] Y. Li, X. Wang, R. Sun *et al.*, "Trustiness-based hierarchical decentralized federated learning," *Knowl. Based Syst.*, p. 110763, 2023.

[43] Y.-R. Yang, K. Wang, and W.-J. Li, "FedREP: A byzantine-robust, communication-efficient and privacy-preserving framework for federated learning," *arXiv preprint arXiv:2303.05206*, 2023.

[44] M. Xhemrishi, J. Östman, A. Wachter-Zeh *et al.*, "FedGT: Identification of malicious clients in federated learning with secure aggregation," *IEEE Trans. Inf. Forensics Security*, vol. 20, pp. 2577–2592, 2025.

[45] D. Boneh, C. Gentry, B. Lynn *et al.*, "Aggregate and verifiably encrypted signatures from bilinear maps," in *International conference on the theory and applications of cryptographic techniques*, 2003, pp. 416–432.

[46] J. Xu, S.-L. Huang, L. Song, and T. Lan, "Byzantine-robust federated learning through collaborative malicious gradient filtering," in *Proc. IEEE 42th Int. Conf. Distrib. Comput. Syst. (ICDCS)*. IEEE, 2022, pp. 1223–1235.

[47] S. P. Karimireddy, L. He, and M. Jaggi, "Learning from history for byzantine robust optimization," in *Proc. Int. Conf. Mach. Learn. (ICML)*. PMLR, 2021, pp. 5311–5319.

[48] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Journal of Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.

[49] E. Shi, H. Chan, E. Rieffel *et al.*, "Privacy-preserving aggregation of time-series data," in *Annual Network & Distributed System Security Symposium (NDSS)*. Internet Society., 2011.

[50] B. Zhao, K. R. Mopuri, and H. Bilen, "iDLG: Improved deep leakage from gradients," *arXiv preprint arXiv:2001.02610*, 2020.

[51] Y. LeCun, L. Bottou, Y. Bengio *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[52] L. Li, W. Xu, T. Chen, G. B. Giannakis, and Q. Ling, "RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets," in *Proc. of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 1544–1551.

[53] C. Xie, O. Koyejo, and I. Gupta, "Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation," in *Proc. of The 35th Uncertainty in Artificial Intelligence Conference*, vol. 115. PMLR, 22–25 Jul 2020, pp. 261–270.

[54] V. Shejwalkar and A. Houmansadr, "Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*. The Internet Society, 2021.

[55] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[56] L. Lei, C. Ju, J. Chen, and M. I. Jordan, "Non-convex finite-sum optimization via scsg methods," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

APPENDIX A
SECURITY AND CONVERGENCE ANALYSIS

A. Proof for Theorem 1

Theorem 1. *Let n be the number of clients, k be the number of clusters, l be the number of members in each cluster, t be the Shamir threshold, and q_1 be the number of type I attackers. Thus, given the security parameter ξ, the constraint on k can be expressed as:*

$$k < \frac{2^{-\xi}}{\sum_{i=t}^{min\{q_1,l-1\}} \frac{q_1!(n-q_1)!l!(n-l)!}{i!(q_1-i)!(l-i)!(n-q_1-l+i)!n!}}$$

Proof. **Case 1:** $q_1 < t$. In this case, the number of malicious clients in a cluster cannot exceed t, since there are fewer than t malicious clients in total. Therefore, the probability that a cluster contains more than t malicious clients is zero:

$$\Pr(X_i \geq t) = 0$$

Thus, for this case, the privacy constraint is trivially satisfied, and no further constraint on k is required.

Case 2: $q_1 \geq t$. To prove this, we begin by defining the number of malicious clients in a single cluster. Suppose that $X_i \sim H(n, q_1, l)$ represents the number of malicious clients in a cluster $\mathcal{C}_i$, where X_i follows a hypergeometric distribution with parameters n, q_1, l. The probability that a cluster contains more than t malicious clients (while excluding the case where all clients are malicious) is given by:

$$\Pr(t \leq X_i < l) = \sum_{i=t}^{min\{l-1,q_1\}} \frac{\binom{q_1}{i}\binom{n-q_1}{l-i}}{\binom{n}{l}}$$

We now expand the binomial coefficients for the hypergeometric distribution. After simplifying the expression, we get:

$$\Pr(t \leq X_i < l) = \sum_{i=t}^{min\{l-1,q_1\}} \frac{q_1!(n-q_1)!l!(n-l)!}{i!(q_1-i)!(l-i)!(n-q_1-l+i)!n!}$$

Now, let A represent the event that at least one cluster contains more than t malicious clients (while excluding the case where all clients are malicious). In this case, the corrupted

clients can collaboratively infer the model privacy of the remaining honest clients. Assuming the groups are independent, the probability that at least one cluster contains more than t malicious clients is:

$$P(A) \leq k \cdot \Pr(t \leq X_i < l)$$

To ensure that the probability of compromising privacy is less than $2^{-\xi}$, we obtain the condition:

$$k \cdot \Pr(t \leq X_i < l) < 2^{-\xi}$$

This condition leads to the following constraint on k:

$$k < \frac{2^{-\xi}}{\sum_{i=t}^{min\{l-1,q_1\}} \frac{q_1!(n-q_1)!l!(n-l)!}{i!(q_1-i)!(l-i)!(n-q_1-l+i)!n!}}$$

In this case, the probability of the privacy security of the scheme being compromised at this point is given by:

$$\Pr(R_{success} = 1) \leq 2^{-\xi}$$

□

B. Proof for Theorem 2

Theorem 2 (Local Model Privacy Under Type I Attackers)**.** *There exists a PPT simulator SIM such that for all PPT adversaries $\mathcal{A}_\mathcal{C}$, all λ, t, $\mathcal{U}$, x_u, and $\mathcal{C}$ such that $\mathcal{C} \subseteq \mathcal{U} \cup \mathcal{S}$ and $|\mathcal{C} \setminus \mathcal{S}| < t$, the output of SIM is computationally indistinguishable from the output of* $\mathsf{REAL}_\mathcal{C}^{\mathcal{U},t,\lambda}$*:*

$$\mathsf{REAL}_\mathcal{C}^{\mathcal{U},t,\lambda}(\mathcal{A}_\mathcal{C}, x_{\mathcal{U}\setminus\mathcal{C}}) \approx_c SIM_\mathcal{C}^{\mathcal{U},t,\lambda,\mathsf{IDEAL}(t,x_u)_{u\in\mathcal{U}\setminus\mathcal{C}}}(\mathcal{A}_\mathcal{C})$$

Proof. The proof is based on a standard hybrid argument. We define a series of subsequent modifications to the real execution of our protocol, such that the views of corrupt parties in any two subsequent executions are computationally indistinguishable.

For fixed n, t, λ and a set $\mathcal{C}$ of corrupt parties, $\mathcal{A}_\mathcal{C}$ denotes the PPT adversary algorithm that represents the "next-message" function of parties in $\mathcal{C}$, i.e. $\mathcal{A}_\mathcal{C}$ outputs the message for party $c \in \mathcal{C}$, and also chooses the honest clients to abort due to failure in a round.

Hyb_0: This random variable represents the combined views of all parties in $\mathcal{C}$ in the real execution REAL of our protocol.

Hyb_1: In this hybrid, SIM runs a full execution of the protocol with $\mathcal{A}_\mathcal{C}$, and simulates LKH-PRF as a random oracle (using a dynamically generated table), PKI and the rest of the setup phase. As a result, the view of the adversary is the same as the previous hybrid.

Hyb_2: This hybrid is identical to H_1, except additionally, SIM substitutes all the encrypted shares sent between pairs of honest clients with encryptions of 0 (in Round 1 - Sharing key).

Note that, we assume the encryption scheme used for encrypting $u_{u,v}$ has IND-CPA security, which guarantees this hybrid is indistinguishable from the previous one.

Hyb_3: This hybrid is identical to H_2 except that SIM aborts if $\mathcal{A}_\mathcal{C}$ queries the random oracle LKH-PRF on input s_u for some honest user u before the adversary receives the responses from honest clients in Round 3 - Unmasking. Because s_u is information theoretically hidden from $\mathcal{A}_\mathcal{C}$, SIM will only abort if $\mathcal{A}_\mathcal{C}$ can make a correct guess of s_u, which is of negligible probability.

Hyb_4: This hybrid is identical to H_3 except that the values of $\mathbf{y}_u$ computed by SIM on behalf of the honest clients and sent to $\mathcal{A}_\mathcal{C}$ (in Round 2 Masked model update collection), are substituted with uniformly sampled values, and the output of some random oracle queries for the LKH-PRF, i.e. $\mathbf{mask}_{s_u}$, is modified to ensure consistency. That is, $\mathbf{mask}_{s_u} \leftarrow \mathbf{y}_u - \mathbf{x}_u$. Because $\mathcal{A}_\mathcal{C}$ cannot query the random oracle on input s_u, the value $\mathbf{y}_u$ looks randomly for the adversary. So the view of the adversary in this hybrid is statistically indistinguishable from the previous one.

Hyb_5: This hybrid is identical to H_4 except that for all honest clients, SIM programs the random oracle to set the output of LKH-PRF as $\mathbf{mask}_{s_u} \leftarrow \mathbf{y}_u - \mathbf{r}_u$, where $\mathbf{r}_u$ is a set of randomly chosen values satisfying $\sum_{u\in\mathcal{U}\setminus\mathcal{C}} \mathbf{x}_u = \sum_{u\in\mathcal{U}\setminus\mathcal{C}} \mathbf{r}_u$. Since $\mathcal{A}_\mathcal{C}$ cannot query the random oracle for LKH-PRF on s_u of honest clients, this hybrid is indistinguishable from the previous one.

Hyb_6: This hybrid is identical to H_5 except that SIM does not receive the inputs of the honest clients, but makes a query to the functionality $\mathsf{IDEAL}(t, x_u)_{u\in\mathcal{U}\setminus\mathcal{C}}$ in Round 3 Unmasking. This modification does not change the view seen by the adversary, and so this hybrid is indistinguishable from the previous one. In addition, this hybrid does not use inputs of the honest parties, and this concludes the proof.

□

C. Proof for Theorem 3

Theorem 3. *Consider the function F that satisfies assumptions 1 and 2. Assume there is a robust aggregation scheme that satisfies assumption 3. Let the number of type II attackers be q_2, and $F^* = min_\theta \nabla(\theta)$. There exists $\eta < \frac{1}{L}(1 - \frac{\sqrt{\delta}}{4} - \frac{q_2}{2n})$ for which, after R rounds, the following condition holds:*

$$\frac{1}{R}\sum_{r=0}^{R-1} \mathbb{E}\|\nabla F(\theta_r)\|^2 \leq \frac{(F(\theta_0) - F^*)}{M\eta R} + \frac{L\eta}{M}\Delta_3 + \frac{1}{M}\Delta_4 \tag{35}$$

where $M = 1 - \frac{\sqrt{\delta}}{4} - \frac{q_2}{2n} - L\eta, \Delta_4 = \frac{8c\sqrt{\delta}}{l}(\sigma_c^2 + \sigma_g^2) + \frac{nq_2\sigma_g^2}{2(n-q_2)^2}, \Delta_3 = 2\Delta_1 + 4\Delta_2 + 12\frac{n-q_2-(1-\beta)kl}{(n-q_2-1)(1-\beta)kl}(\sigma_c^2 + \sigma_g^2 + \Delta_2), \Delta_1 = \frac{8c\delta}{l}(\sigma_c^2 + \sigma_g^2) + b^2, \Delta_2 = \frac{q_2^2\sigma_g^2}{(n-q_2)^2} + \frac{2\sigma_c^2}{n-q_2}$.

Proof. Let the remaining benign clients be $\mathcal{U}_b$. Define $\bar{\mathbf{w}}_r^{(\mathcal{U}_b)}$ as the average model update of the benign clients. Note that the number of clients in the benign clusters is less than or equal to the number of benign clients ($|\{u_i\}_{u_i\in\mathcal{G}_g}| \leq |\mathcal{U}_b|$), since the Byzantine clusters may also contain benign clients.

For $\forall u_i, u_j \in \mathcal{U}_b$, we have

$$\begin{aligned}
&\mathbb{E}\left[\|\mathbf{w}_r^{(u_i)} - \mathbf{w}_r^{(u_j)}\|^2\right] \\
&= \mathbb{E}[\|\mathbf{w}_r^{(u_i)} - \nabla F_i(\theta_r) + \nabla F_i(\theta_r) - \nabla F(\theta_r) \\
&+ \nabla F(\theta_r) - \nabla F_j(\theta_r) + \nabla F_j(\theta_r) - \mathbf{w}_r^{(u_j)}\|^2] \\
&\overset{(a)}{\leq} 4(\mathbb{E}\left[\|\mathbf{w}_r^{(u_i)} - \nabla F_i(\theta_r)\|^2\right] + \mathbb{E}\left[\|\nabla F_i(\theta_r) - \nabla F(\theta_r)\|^2\right] \\
&+ \mathbb{E}\left[\|\nabla F(\theta_r) - \nabla F_j(\theta_r)\|^2\right] + \mathbb{E}\left[\|\nabla F_j(\theta_r) - \mathbf{w}_r^{(u_j)}\|^2\right]) \\
&\overset{(b)}{\leq} 8(\sigma_c^2 + \sigma_g^2)
\end{aligned} \tag{36}$$

where (a) follows Reddi et al. [55] (A.4, Lemma 6), while (b) is based on Assumptions 2.

We first prove that the difference between the robust aggregation update and the benign cluster average update is bounded:

$$\begin{aligned}
&\mathbb{E}[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}}) - \bar{\mathbf{w}}_r\|^2] \\
&\overset{(a)}{=} \mathbb{E}[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}}) - \bar{\mathbf{w}}_r\|]^2 \\
&+ \text{Var}\left[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}})\|\right] \\
&\overset{(b)}{\leq} c\delta \sup_{\mathcal{C}_i, \mathcal{C}_j \in \mathcal{G}_g} \mathbb{E}\left[\|\mathbf{w}_r^{(\mathcal{C}_i)} - \mathbf{w}_r^{(\mathcal{C}_j)}\|^2\right] + b^2 \\
&= \frac{c\delta}{l^2} \sup_{\mathcal{C}_i, \mathcal{C}_j \in \mathcal{G}_g} \mathbb{E}\left[\|\sum_{u_i \in \mathcal{C}_i} \mathbf{w}_r^{(u_i)} - \sum_{u_j \in \mathcal{C}_j} \mathbf{w}_r^{(u_j)}\|^2\right] + b^2 \\
&\overset{(c)}{\leq} \underbrace{\frac{8c\delta}{l}(\sigma_c^2 + \sigma_g^2) + b^2}_{\Delta_1}
\end{aligned} \tag{37}$$

where (a) follows from the basic relationship between expectation and variance, (b) is based on Assumption 3, and (c) uses (36) and Reddi et al. [55] (A.4, Lemma 6).

Next, we prove that the difference between the benign cluster average update and the global true update is bounded:

$$\begin{aligned}
&\mathbb{E}\left[\|\bar{\mathbf{w}}_r - \nabla F(\theta_r)\|^2\right] \\
&= \mathbb{E}\left[\|\bar{\mathbf{w}}_r - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)} + \bar{\mathbf{w}}_r^{(\mathcal{U}_b)} - \nabla F(\theta_r)\|^2\right] \\
&\leq 2\mathbb{E}\left[\|\bar{\mathbf{w}}_r - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)}\|^2\right] + 2\mathbb{E}\left[\|\bar{\mathbf{w}}_r^{(\mathcal{U}_b)} - \nabla F(\theta_r)\|^2\right] \\
&\overset{(a)}{\leq} 2\mathbb{E}\left[\|\bar{\mathbf{w}}_r - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)}\|^2\right] + 2\underbrace{\frac{q_2^2\sigma_g^2}{(n-q_2)^2} + \frac{2\sigma_c^2}{n-q_2}}_{\Delta_2}
\end{aligned} \tag{38}$$

where (a) follows Xu et al. [46] (Lemma 1).

We further prove that the difference between the benign cluster average update and the average update of all benign clients is bounded:

$$\begin{aligned}
&\mathbb{E}\left[\|\bar{\mathbf{w}}_r - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)}\|^2\right] \\
&= \mathbb{E}\left[\|\frac{1}{(1-\beta)kl}\sum_{u_i \in \mathcal{G}_g}(\mathbf{w}_r^{(u_i)} - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)})\|^2\right] \\
&\overset{(a)}{\leq} \frac{n - q_2 - (1-\beta)kl}{(n-q_2-1)(1-\beta)kl} \cdot \frac{1}{n-q_2}\sum_{u_i \in \mathcal{U}_b}\mathbb{E}\left[\|\mathbf{w}_r^{(u_i)} - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)}\|^2\right]
\end{aligned} \tag{39}$$

where (a) follows Lei et al. [56] (Lemma A.1), because the clustering is completely random, $\{u_i\}_{u_i \in \mathcal{G}_g}$ can be considered a random sample from the set $\mathcal{U}_b$.

Note that

$$\begin{aligned}
&\mathbb{E}\left[\|\mathbf{w}_r^{(u_i)} - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)}\|^2\right] \\
&= \mathbb{E}[\|\mathbf{w}_r^{(u_i)} - \nabla F_i(\theta_r) + \nabla F_i(\theta_r) \\
&- \nabla F(\theta_r) + \nabla F(\theta_r) - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)}\|^2] \\
&\overset{(a)}{\leq} 3\mathbb{E}\left[\|\mathbf{w}_r^{(u_i)} - \nabla F_i(\theta_r)\|^2\right] + 3\mathbb{E}\left[\|\nabla F_i(\theta_r) - \nabla F(\theta_r)\|^2\right] \\
&+ 3\mathbb{E}\left[\nabla F(\theta_r) - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)}\|^2\right] \\
&\overset{(b)}{\leq} 3\sigma_c^2 + 3\sigma_g^2 + 3\Delta_2
\end{aligned} \tag{40}$$

where (a) refers to Reddi et al. [55] (A.4, Lemma 6), and (b) is based on (38) and Assumption 2.

So, we can get from (39) and (40)

$$\begin{aligned}
&\mathbb{E}\left[\|\bar{\mathbf{w}}_r - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)}\|^2\right] \\
&\leq 3\frac{n - q_2 - (1-\beta)kl}{(n-q_2-1)(1-\beta)kl}(\sigma_c^2 + \sigma_g^2 + \Delta_2)
\end{aligned} \tag{41}$$

Finally, we aim to prove that the difference between the robust aggregation update and the global true update is bounded:

$$\begin{aligned}
&\mathbb{E}_r[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}}) - \nabla F(\theta_r)\|^2] \\
&= \mathbb{E}_r[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}}) - \bar{\mathbf{w}}_r + \bar{\mathbf{w}}_r - \nabla F(\theta_r)\|^2] \\
&\overset{(a)}{\leq} 2\mathbb{E}_r[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}}) - \bar{\mathbf{w}}_r\|^2] \\
&+ 2\mathbb{E}_r[\|\bar{\mathbf{w}}_r - \nabla F(\theta_r)\|^2] \\
&\overset{(b)}{\leq} \underbrace{2\Delta_1 + 4\Delta_2 + 12\frac{n - q_2 - (1-\beta)kl}{(n-q_2-1)(1-\beta)kl}(\sigma_c^2 + \sigma_g^2 + \Delta_2)}_{\Delta_3}
\end{aligned} \tag{42}$$

where (a) follows Reddi et al. [55] (A.4, Lemma 6), while (b) utilizes (37), (38), and (41).

We will now proceed with the derivation of the convergence

$$\begin{aligned}
&\mathbb{E}_r[F(\theta_{r+1})] - F(\theta_r) \\
&\overset{(a)}{\leq} -\langle \nabla F(\theta_r), \mathbb{E}_r[\theta_{r+1} - \theta_r]\rangle + \frac{L}{2}\mathbb{E}_r[\|\theta_r - \theta_{r+1}\|^2] \\
&= -\eta\langle \nabla F(\theta_r), \mathbb{E}_r\left[\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}})\right]\rangle \\
&+ \frac{L\eta^2}{2}\mathbb{E}_r\left[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}})\|^2\right] \\
&= -\eta\langle \nabla F(\theta_r), \mathbb{E}_r\left[\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}})\right] - \nabla F(\theta_r) + \\
&\nabla F(\theta_r)\rangle + \frac{L\eta^2}{2}\mathbb{E}_r[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}}) \\
&- \nabla F(\theta_r) + \nabla F(\theta_r)\|^2] \\
&\overset{(b)}{\leq} -\eta\langle \nabla F(\theta_r), \mathbb{E}_r\left[\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}})] - \nabla F(\theta_r)\right]\rangle \\
&+ L\eta^2\mathbb{E}_r\left[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi \in \mathcal{C}}) - \nabla F(\theta_r)\|^2\right] \\
&+ (L\eta^2 - \eta)\|\nabla F(\theta_r)\|^2
\end{aligned} \tag{43}$$

where (a) applies the property of $L-Lipschitz$ continuity, and (b) relies on Reddi et al. [55] (A.4, Lemma 6).

Note that

$$
\begin{aligned}
&-\langle\nabla F(\theta_r), \mathbb{E}_r\left[\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}})] - \nabla F(\theta_r)\right]\rangle \\
&= -\langle\nabla F(\theta_r), \mathbb{E}_r\left[\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}})] - \nabla F(\theta_r)\right]\rangle \\
&= -\langle\nabla F(\theta_r), \mathbb{E}_r\left[\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}})] - \bar{\mathbf{w}}_r\right]\rangle \\
&- \langle\nabla F(\theta_r), \mathbb{E}_r\left[\bar{\mathbf{w}}_r - \bar{\mathbf{w}}_r^{(\mathcal{U}_b)}\right]\rangle \\
&- \langle\nabla F(\theta_r), \mathbb{E}_r\left[\bar{\mathbf{w}}_r^{(\mathcal{U}_b)} - \nabla F(\theta_r)\right]\rangle \\
&\overset{(a)}{=} -\langle\nabla F(\theta_r), \mathbb{E}_r\left[\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}})] - \bar{\mathbf{w}}_r\right]\rangle \\
&- \langle\nabla F(\theta_r), \mathbb{E}_r\left[\bar{\mathbf{w}}_r^{(\mathcal{U}_b)} - \nabla F(\theta_r)\right]\rangle
\end{aligned}
\tag{44}
$$

where (a) is due to the fact that $\{u_i\}_{u_i\in\mathcal{G}_g}$ is a random sample from $\mathcal{U}_b$, and the overall expectation equals the sample expectation, i.e., $\mathbb{E}_r[\bar{\mathbf{w}}_r] = \mathbb{E}_r[\bar{\mathbf{w}}_r^{(\mathcal{U}_b)}]$.

We will first analyze the first term:

$$
\begin{aligned}
&-\langle\nabla F(\theta_r), \mathbb{E}_r\left[\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}})] - \bar{\mathbf{w}}_r\right]\rangle \\
&\overset{(a)}{\leq} \|\nabla F(\theta_r)\| \cdot \mathbb{E}_r\left[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}})] - \bar{\mathbf{w}}_r\|\right] \\
&\overset{(b)}{\leq} \frac{1}{2}\Big(\frac{2}{\sqrt{\delta}}\mathbb{E}_r\left[\|\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}})] - \bar{\mathbf{w}}_r\|\right]^2 \\
&+ \frac{\sqrt{\delta}}{2}\|\nabla F(\theta_r)\|^2\Big) \\
&\overset{(c)}{\leq} \frac{\sqrt{\delta}}{4}\|\nabla F(\theta_r)\|^2 + \frac{8c\sqrt{\delta}}{l}(\sigma_c^2 + \sigma_g^2)
\end{aligned}
\tag{45}
$$

where (a) applies the Cauchy-Schwarz inequality, (b) utilizes the AM-GM inequality, and (c) follows (37).

Next, we analyze the second term:

$$
\begin{aligned}
&-\langle\nabla F(\theta_r), \mathbb{E}_r\left[\bar{\mathbf{w}}_r^{(\mathcal{U}_b)} - \nabla F(\theta_r)\right]\rangle \\
&\overset{(a)}{\leq} \|\nabla F(\theta_r)\| \cdot \mathbb{E}_r\left[\|\bar{\mathbf{w}}_r^{(\mathcal{U}_b)} - \nabla F(\theta_r)\|\right] \\
&\overset{(b)}{\leq} \frac{1}{2}\Big(\frac{q_2}{n}\|\nabla F(\theta_r)\|^2 + \frac{n}{q_2}\mathbb{E}_r\left[\|\bar{\mathbf{w}}_r^{(\mathcal{U}_b)} - \nabla F(\theta_r)\|\right]^2\Big) \\
&\overset{(c)}{\leq} \frac{q_2}{2n}\|\nabla F(\theta_r)\|^2 + \frac{nq_2\sigma_g^2}{2(n-q_2)^2}
\end{aligned}
\tag{46}
$$

where (a) applies the Cauchy-Schwarz inequality, (b) utilizes the AM-GM inequality, and (c) is based on the derivation from Xu et al. [46] (Lemma 1).

Combining (44), (45), and (46), we have

$$
\begin{aligned}
&-\langle\nabla F(\theta_r), \mathbb{E}_r\left[\textbf{RobustAgg}(\{\mathbf{w}_r^{(\mathcal{C}_\varphi)}\}_{\mathcal{C}_\varphi\in\mathcal{C}})] - \nabla F(\theta_r)\right]\rangle \\
&\leq \Big(\frac{\sqrt{\delta}}{4} + \frac{q_2}{2n}\Big)\|\nabla F(\theta_r)\|^2 + \underbrace{\frac{8c\sqrt{\delta}}{l}(\sigma_c^2 + \sigma_g^2) + \frac{nq_2\sigma_g^2}{2(n-q_2)^2}}_{\Delta_4}
\end{aligned}
\tag{47}
$$

Substituting (47) into (43) and reorganizing, we obtain

$$
\begin{aligned}
&\eta\Big(1 - \frac{\sqrt{\delta}}{4} - \frac{q_2}{2n} - L\eta\Big)\|\nabla F(\theta_r)\|^2 \leq \mathbb{E}_r[F(\theta_r) - F(\theta_{r+1})] + \\
&+ L\eta^2\Delta_3 + \eta\Delta_4
\end{aligned}
\tag{48}
$$

Assuming $\eta < \frac{1}{L}(1 - \frac{\sqrt{\delta}}{4} - \frac{q_2}{2n})$, we can get $M = 1 - \frac{\sqrt{\delta}}{4} - \frac{q_2}{2n} - L\eta > 0$, sum over the federated learning rounds from $0 \to R$, and we obtain

$$
\frac{1}{R}\sum_{r=0}^{R-1}\mathbb{E}\|\nabla F(\theta_r)\|^2 \leq \frac{(F(\theta_0) - F^*)}{M\eta R} + \frac{L\eta}{M}\Delta_3 + \frac{1}{M}\Delta_4 \tag{49}
$$

□

APPENDIX B
EVALUATION

A. Supplement: Multi-round Detection

Figure 13 illustrates the frequency distribution of each client u_i being detected as anomalous $H_{r_t}(u_i)$ after $r_t = 35$ rounds of federated training on the CIFAR-10 dataset. The experimental settings are consistent with those described in Section VII-B of the main text. Red dots represent clients labeled as anomalous, while blue dots represent normal clients. Under different poisoning attack scenarios, the detection results for anomalous clients exhibit clear distribution patterns, further validating the effectiveness of our detection scheme.

Under the label flipping attack (Figure 13(a)), the anomalous clients are more scattered. Some malicious clients have relatively little impact on the model during training, and thus are marked as anomalous less frequently compared to other attack types. Nevertheless, even in such cases, our scheme can still effectively identify the majority of malicious clients (11 out of the 12 predefined malicious clients are correctly detected, and the remaining one can be eliminated in subsequent rounds of detection).

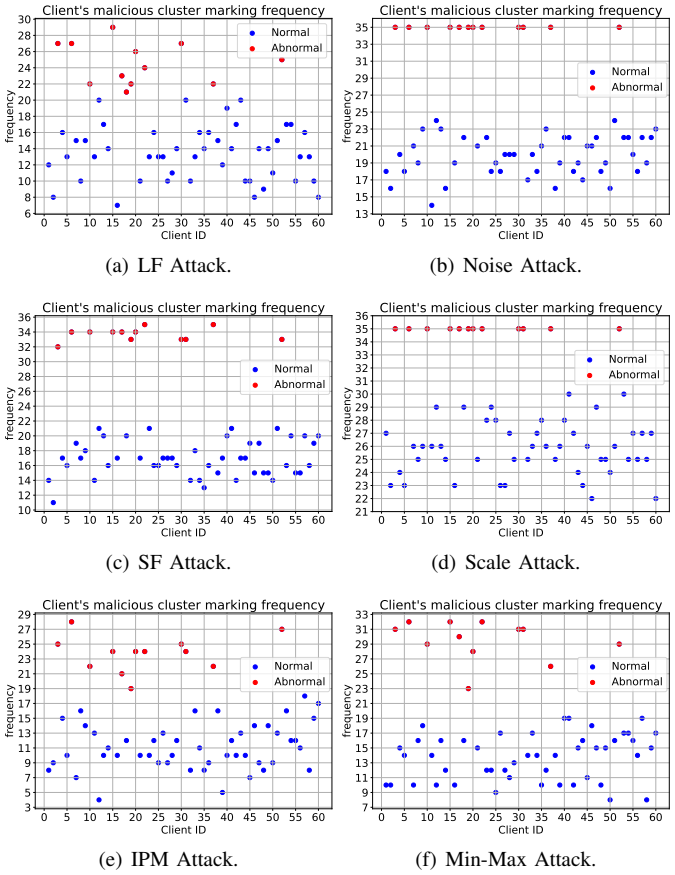
In the noise attack (Figure 13(b)) and the scale attack (Figure 13(d)), the attacks have a more significant impact on the model training, leading to a more concentrated distribution of anomalous clients. Our scheme accurately identifies these malicious clients while maintaining detection stability for benign clients, thus avoiding false positives.

Under the sign flipping attack (Figure 13(c)), the IPM attack (Figure 13(e)), and the Min-Max attack (Figure 13(f), the frequency distribution of anomalous clients is relatively concentrated, and all malicious clients are successfully detected (F1-score = 1.000).

B. Evaluation: Further Results

A Byzantine-robust, cluster-based privacy-preserving federated learning scheme can be extended to a decentralized federated learning setting. In the decentralized federated learning setting, In this paradigm, the security analysis against Type-I attackers remains consistent with the previous analysis, as the multiple leaders collectively assume the role of the server. For Type-II attackers, we consider two scenarios:

1. Non-leader clients: In this case, malicious attacks can still be countered using the previously established defense



(a) LF Attack. (b) Noise Attack.

(c) SF Attack. (d) Scale Attack.

(e) IPM Attack. (f) Min-Max Attack.

Fig. 13. Multi-round federated learning anomaly detection results on the CIFAR-10 dataset

mechanisms. However, the trust in this scenario does not stem from a trusted dataset collected by the server but rather from the client's own local dataset.

2. Leader clients: Leaders may attempt to send entirely incorrect model updates to the clients. Fortunately, the baffle mechanism ensures that if all the received model updates exhibit extremely low similarity to the client's own updates, the client can reject these updates.

We present the evaluation of the ClusterGuard federated learning scheme in a decentralized setting with 20 clients. The training configuration, as well as the model and dataset settings, align with the default setup described earlier. We use the average model accuracy of honest clients to estimate the model performance. We consider two situations: the first case is the attack from non-leader clients, and the second case is the attack from leader clients.

First case: the attack setup is the same as described earlier, where we still consider the different attack strategies mentioned above. The number of adversaries is set to 20% of the total clients.

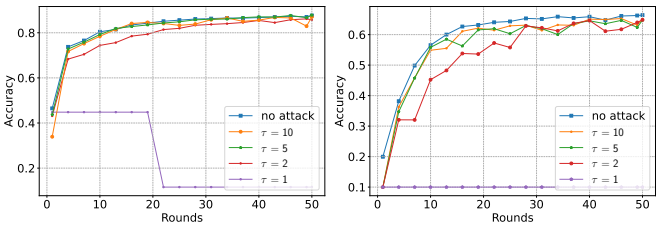
Table VI presents a comprehensive evaluation of ClusterGuard in comparison to other robust aggregation methods (including Median, Krum, Share, and Bulyan) across various malicious attack strategies, including label flipping, noise injection, sign flipping, and scaling, on the MNIST, Fashion MNIST, and CIFAR-10 datasets. The experimental results

TABLE VI
ACCURACY(%) COMPARISON OF CLUSTERGUARD AND OTHER SCHEMES UNDER MALICIOUS ATTACKS IN DECENTRALIZED SETTING

Scheme	LF Attack	Noise Attack	SF Attack	Scale Attack	IPM Attack	Min-Max Attack
MNIST						
Krum	63.28	66.11	60.75	72.43	46.50	52.07
Median	69.30	79.37	68.21	85.03	69.98	79.40
Share	73.59	**87.14**	29.97	34.95	83.78	**83.67**
Bulyan	69.35	68.77	69.41	77.69	42.12	66.38
ClusterGuard	**86.96**	85.54	**82.03**	**85.11**	**85.68**	82.88
Fasion-MNIST						
Krum	69.32	67.32	65.61	69.92	69.51	71.89
Median	83.43	86.73	79.34	**87.96**	84.03	83.50
Share	73.39	86.85	51.60	26.94	80.00	84.18
Bulyan	84.17	85.24	85.23	84.85	65.68	78.75
ClusterGuard	**85.11**	**86.91**	**85.29**	85.28	**84.78**	**85.68**
CIFAR-10						
Krum	29.60	37.83	36.66	36.78	10.00	34.90
Median	41.85	64.19	53.84	**67.89**	59.70	59.85
Share	43.52	64.38	62.58	11.65	51.74	56.12
Bulyan	14.91	58.35	60.19	60.88	26.21	41.51
ClusterGuard	**64.67**	**65.30**	**63.44**	64.33	**65.54**	**64.12**

consistently demonstrate that ClusterGuard outperforms the other defense strategies across all three datasets. Methods such as Krum and Share are susceptible to carefully crafted attacks, whereas ClusterGuard maintains consistently high accuracy and resilience. Moreover, in contrast to Median and Bulyan, which are limited to simpler tasks such as MNIST digit classification, ClusterGuard demonstrates superior performance on the more complex image classification task of CIFAR-10, where other methods show significant performance degradation. These findings establish ClusterGuard as a robust and reliable solution for decentralized federated learning, capable of defending against a wide range of diverse and sophisticated attack strategies.

Second case: we utilize the Random Attack method to simulate the behavior of adversarial clients. Byzantine clients send gradients with randomized values, generated from a multi-dimensional Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. In our experiments, we set $\mu = 0$ and $\sigma = 0.5$ to simulate random attacks. Let τ denote the attack interval for all leader clients. In practice, such attacks are challenging to sustain, as the leader selection process is fully random, based on a VRF-based clustering mechanism.



(a) Fasion-MNIST. (b) CIFAR-10.

Fig. 14. The impact of different leader client attack intervals on model accuracy.

The experimental results demonstrate that our algorithm

remains robust under certain attack frequencies, achieving satisfactory final accuracy by utilizing the "baffle mechanism". For instance, when $\tau = 10$ or $\tau = 5$ (less frequent attacks), the model achieves comparable accuracy to scenarios without attacks, showcasing the effectiveness of our proposed scheme. In contrast, the case of $\tau = 1$, where attacks occur in every round, represents an extreme situation. Here, training stagnates entirely, and accuracy fails to improve. However, this extreme scenario is highly improbable under a random clustering mechanism, as adversarial clients are unlikely to dominate the leader role consistently. This highlights the robustness of the proposed approach in realistic settings, ensuring stable performance despite adversarial interference.