SIREN$^+$: Robust Federated Learning with Proactive Alarming and Differential Privacy

Hanxi Guo, Hao Wang, Tao Song, *Member, IEEE,*
Yang Hua, Ruhui Ma, Xiulang Jin, Zhengui Xue, and Haibing Guan, *Member, IEEE*

Abstract—Federated learning (FL), an emerging machine learning paradigm that trains a global model across distributed clients without violating data privacy, has recently attracted significant attention. However, FL's distributed nature and iterative training extensively increase the attacking surface for Byzantine and inference attacks. Existing FL defense methods can hardly protect FL from both Byzantine and inference attacks due to their fundamental conflicts. The noise injected to defend against inference attacks interferes with model weights and training data, obscuring model analysis that Byzantine-robust methods utilize to detect attacks. Besides, the practicability of existing Byzantine-robust methods is limited since they heavily rely on model analysis.
In this paper, we present SIREN$^+$, a new robust FL system that defends against a wide spectrum of Byzantine attacks and inference attacks by jointly utilizing a proactive alarming mechanism and local differential privacy (LDP). The proactive alarming mechanism orchestrates clients and the FL server to collaboratively detect attacks using distributed alarms, which are free from the noise interference injected by LDP. Compared with the state-of-the-art defense methods, SIREN$^+$ can protect FL from Byzantine and inference attacks from a higher proportion of malicious clients in the system while keeping the global model performing normally. Extensive experiments with diverse settings and attacks on real-world datasets show that SIREN$^+$ outperforms existing defense methods when attacked by Byzantine and inference attacks.

Index Terms—Federated Learning, Byzantine-robust, Attack-agnostic Defense System, Differential Privacy

1 INTRODUCTION

DATA security and privacy are gaining increasing attention with the prevalence of machine learning. Many countries enacted laws [2] to regularize the data collection of companies and organizations, leading to "data silos". To adapt to such circumstances, federated learning (FL) [3, 4] is proposed. Unlike traditional machine learning that centralizes data to a cluster of servers for training [5, 6, 7], FL trains the model via iteratively aggregating the models or model updates that are uploaded by a swarm of loosely connected devices. Since it does not directly utilize the raw data, the data security and privacy of the participants are guaranteed. With such merit, FL has an extensive range of applications [8, 9], including training language models [10], health tracking [11, 12], and fintech [13] with privacy-sensitive data such as user keystrokes, photos, and geo-locations.

However, due to its distributed and iterative nature, FL is vulnerable to various attacks from compromised clients [14]. It is hardly possible to determine the concrete number of malicious clients since the unpredictable behaviors of attackers. Besides, the local data on clients are typically non-independent and identically distributed (non-IID). Such data skews across participating devices aggravate the divergence between local models, further obfuscating the boundary between malicious clients and benign ones.

As a consequence, FL's intrinsic vulnerabilities have induced a rich literature of studies on attacking methodologies, *e.g.*, Byzantine attacks [15] and inference attacks [16, 17]. Byzantine attacks attempt to corrupt the federated model by smuggling malicious model updates through compromised or fake clients [18, 19, 20, 21, 22, 23, 24, 25]. Inference attacks aim to extract sensitive information of clients' private data from local updates and global weights in FL [16, 17, 26, 27, 28].

To defend against Byzantine attacks, researchers have developed various Byzantine-robust FL frameworks using *model analysis*. Some of them [29, 30, 31, 32, 33, 34, 35, 36, 37] detect malicious clients by analyzing model gradients and measuring the difference between clients' model updates on the server. While others [38, 39] try to alleviate the attacks through benign client efforts. To defend against inference attacks, differential privacy (DP) [40] is used to protect FL by carefully injecting noise to model updates. Several studies [10, 40, 41, 42, 43, 44, 45, 46, 47] have already attempted to add elaborately crafted noise to FL's different stages and achieved impressive privacy guarantees while keeping acceptable model performance.

Limitations of existing defenses. Existing studies rarely provide a comprehensive solution to defend against multiple types of attacks while minimizing the potential risks. They leave the following critical concerns unaddressed: *First*, current defenses can hardly defend against both

- *Preliminary results have been presented in ACM SoCC'21 [1]*
- *H. Guo, T. Song (the corresponding author), R. Ma and H. Guan are with the Shanghai Key Laboratory of Scalable Computing and Systems, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (Email: {hanxiguo, songt333, ruhuima, hbguan}@sjtu.edu.cn)*
- *H. Wang is with the Computer Science & Engineering, Louisiana State University, Baton Rouge, LA 70803-2804, USA (Email: haowang@lsu.edu)*
- *Y. Hua and Z. Xue are with the EEECS/ECIT and School of Mathematics and Physics, Queen's University Belfast, Belfast BT7 1NN, UK (Email: {Y.Hua, z.xue}@qub.ac.uk)*
- *X. Jin is with Huawei Technologies Co., Ltd, Hangzhou 310000, China (Email: jinxiulang@huawei.com)*

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3362534

Byzantine and inference attacks due to the fundamental conflicts between existing defense methodologies against Byzantine and inference attacks. The training data and model updates transformed by defensive methods for inference attacks, such as DP, extensively confuse existing Byzantine-robust FL methods, which analyze model weights (or gradients) to detect attacks. *Second*, current defensive methods cannot effectively handle more challenging and realistic scenarios. For example, non-IID data amplifies the divergence in model weights trained by different clients [48], triggering existing methods to drop model updates from benign clients. Common defensive methods also regard benign clients as malicious clients easily when the malicious clients become the majority. *Finally*, existing methods only rely on the detection mechanism either on the FL server or on the clients [38, 39], while benign clients have to passively accept the corrupted global model without any resistance once the global model has been successfully affected by malicious clients. The robustness and self-repairing capability of existing Byzantine-robust aggregation rules are worrying. Though existing work [49] illustrates the limitation of current poisoning attacks, such critical concerns should still be considered from a defender's perspective.

Thus, in this paper, we present SIREN$^+$, a new robust FL system that defends against a wide spectrum of Byzantine attacks and inference attacks by jointly utilizing a proactive alarming mechanism and local DP (LDP). Specifically, the alarming mechanism is proactive and distributed, which enables clients to collaborate with the FL server on attack detection via model accuracy analysis. SIREN$^+$ clients reserve a small partition of the local dataset to test the global model accuracy and trigger alarms if needed, and the FL server jointly analyzes clients' alarms and model testing accuracy to detect attacks. Unlike existing Byzantine-robust methods that only rely on model analysis, the accuracy-based alarming mechanism used by SIREN$^+$ is compatible with LDP. Based on this distributed alarming system, we carefully craft a decision process that detects the intentions of malicious clients and sanitizes the model aggregation effectively.

Extensive experimental results of training a CNN model with the Fashion-MNIST dataset [50] and a ResNet-18 model with the CIFAR-10 dataset [51] under a system of up to 200 clients show that SIREN$^+$ can effectively defend against both untargeted and targeted Byzantine attacks as well as both trained and untrained inference attacks. SIREN$^+$ outperforms current Byzantine-robust aggregation rules (*e.g.*, multi-Krum, FLTrust, Coordinate-wise Median) when facing Byzantine attacks under typical (40% malicious client), extreme (80% malicious client), and real-world (simultaneously multiple attacks) scenarios over both IID and non-IID data distribution. When facing inference attacks, SIREN$^+$ reduces the area under the curve (AUC) of attacker models from ∼70% to ∼50%.

Our contributions. To the best of our knowledge, we are the first to propose a comprehensively robust FL system against both Byzantine attacks and inference attacks via orchestrating both the defenses on the FL server and clients. The main contributions of this paper are summarized as follows:

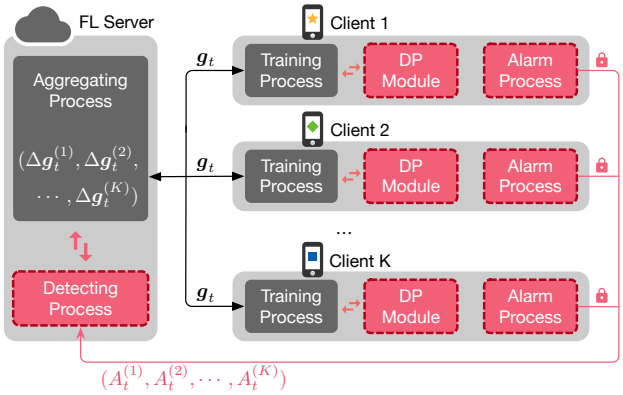


Fig. 1: The architecture of FL. The gray blocks belong to the default FL paradigm, and the red blocks with dotted borders are SIREN$^+$'s components.

TABLE 1: Notations.

Symbol	Meaning
t	the FL communication round index
K	the set of participating clients
S_a	the set of clients with activated alarms
S_s	the set of clients with no alarms, $S_a \cup S_s = K$
S_b	the set of benign clients, $S_b \subseteq K$
$\boldsymbol{g}_t$	the global model at round t
$\boldsymbol{g}_t^{(i)}$	Client i's local model at round t
$\Delta \boldsymbol{g}_t^{(i)}$	the local model update of client i
ω_t	the global model's testing accuracy at round t
$\omega_t^{(i)}$	the local model's testing accuracy of client i
$A_t^{(i)}$	the alarm on client i at round t
$D^{(i)}$	the local training dataset of client i
D_0	the root test dataset of the FL server
$D_0^{(i)}$	the local test dataset of client i
$\alpha^{(i)}$	the aggregation weight of client i
C_c	a user-defined threshold for clients
C_s	a user-defined threshold for the FL server

- We design SIREN$^+$, a comprehensively attack-agnostic defense system that can defend against both Byzantine and inference attacks by jointly utilizing the proactive alarming and LDP mechanisms.
- We enable a collaborative defense mode involving both the FL server end (the decision process) and the client end (the LDP-compatible alarming process) to provide more efficient defense while minimizing the potential damages of various attacks. Our client-triggered design improves the self-protection capability of clients, in turn enhancing the self-repairing ability of the whole FL system.
- We thoroughly evaluate SIREN$^+$ under typical, extreme, and real-world scenarios over both IID and non-IID data distribution under various Byzantine attacks and inference attacks. Experimental results show that SIREN$^+$ can effectively defend against such attacks while achieving SOTA model performance compared with prevailing defensive methods.

2 BACKGROUND

2.1 Federated Learning

Federated Learning (FL) iteratively aggregates model updates from multiple client devices to train a shared global model without violating clients' data privacy. As shown in Fig. 1, in a communication round t, a remote FL server first pushes a global model $\boldsymbol{g}_t$ to client devices and collects model updates $\{\Delta \boldsymbol{g}_t^{(i)} | i \in K\}$ from a set of $|K|$ randomly selected client devices to update the global model from $\boldsymbol{g}_t$ to $\boldsymbol{g}_{t+1}$. The updated global model $\boldsymbol{g}_{t+1}$ is then pushed to client devices for the next round [3, 4]. Clients' local datasets may follow different distributions and are inaccessible to the FL server or other clients. The meanings of all the symbols used in this paper are presented in Table 1.

Specifically, we select $|K|$ client devices to participate in FL every round, and each client device i has a local dataset $D^{(i)}$, $i \in K$. Each participating client trains a local model $\boldsymbol{g}_t^{(i)}$ using its own data with an objective to jointly solve the following optimization problem—minimizing the expected empirical loss $F(\boldsymbol{g})$ on the training data across client devices:

$$\min_{\boldsymbol{g}} F(\boldsymbol{g}) := \min_{\boldsymbol{g}} \sum_{i=1}^{|K|} \mathbb{E}_{D^{(i)} \sim \mathcal{X}^{(i)}}[f(D^{(i)}, \boldsymbol{g})], \tag{1}$$

where $\boldsymbol{g}$ is the global model, $D^{(i)}$ is a set of local training data samples following an unknown distribution $\mathcal{X}^{(i)}$ on client i, and f denotes the local loss function.

The FL server orchestrates participating client devices to jointly solve this optimization problem. In the t-th communication round, each client i initializes its local model with the global model $\boldsymbol{g}_t$ and trains the model locally with gradient descent algorithms: $\boldsymbol{g}_{t+1}^{(i)} := \arg\min_{\boldsymbol{g}} \mathbb{E}_{D^{(i)} \sim \mathcal{X}^{(i)}}[f(D^{(i)}, \boldsymbol{g})]$. When the local training completes, the client i calculates and pushes the local model update $\Delta \boldsymbol{g}_{t+1}^{(i)} := \boldsymbol{g}_{t+1}^{(i)} - \boldsymbol{g}_t$ to the FL server. Typically, the FL server updates the global model using the federated averaging (FEDAVG) algorithm [4]:

$$\boldsymbol{g}_{t+1} \leftarrow \boldsymbol{g}_t + \eta \sum_{i=1}^{|K|} \frac{|D^{(i)}|}{|D|} \Delta \boldsymbol{g}_{t+1}^{(i)}, \tag{2}$$

where η is the learning rate, and $D := \sum_{i=1}^{|K|} D^{(i)}$ denotes the total data samples of the $|K|$ client devices. Then, the FL server sends the new global model $\boldsymbol{g}_{t+1}$ to client devices and starts the next communication round.

2.2 Attack Taxonomy

Due to its distributed and iterative nature, existing FL systems are vulnerable to various types of Byzantine attacks and inference attacks [14].

Byzantine attacks: Byzantine attacks can be divided into three categories: model poisoning attack, data poisoning attack, and the compound of the previous two categories of attacks. In this paper, we use two types of model poisoning attacks, one data poisoning attack, and one compound attack. *Sign-flipping Attack* [18] is the first type of model poisoning attack, where malicious clients train the model as what benign clients do and obtain normal gradients[1] but multiplying the normal gradients by a negative constant when uploading. *Adaptive Attack* [19] is the second type of model poisoning attack, aiming to drive the changing direction of the global model to the reverse of the normal direction through customized optimization. *Label-flipping Attack* [1, 19] is the data poisoning attack that lets the malicious client train with normal images but with flipped labels. These three attacks are all untargeted attacks. *Targeted Model Poisoning* [22] is a compound and targeted attack that aims to extensively degrade model accuracy at specific data categories but perform as a normal model at other categories by manipulating the model parameters.

Inference attacks: Generally, inference attacks can be divided into mainly four types [14], which are *Class Representative Inference Attack* [26], *Membership Inference Attack* (MIA) [16, 27, 52], *Property Inference Attack* [16], and *Input&Label Inference Attack* [17, 28]. Among all the categories, MIA is the most widely used type. MIA trains various attacker models to determine whether a specific sample is used to train the target model. It can violate clients' data privacy under both white-box and black-box settings in FL. Thus, in this paper, we use two types of MIAs - threshold MIA (untrained) and logistic regression MIA (trained) to evaluate the LDP part of SIREN$^+$. Such two MIAs follow the design of the original MIA [52] with modifications based on the findings of *ML-Leaks* [53] to improve the efficiency.

2.3 Common Defenses

This section introduces three prevailing types of defense methods used as the baseline defense schemes in the following experiments for comparison.

Krum [29]. In Krum, the FL server computes the score $s_t^{(i)}$ of each weight update in each communication round, where $s_t^{(i)} = \sum_{i \to j} ||\Delta \boldsymbol{g}_t^{(i)} - \Delta \boldsymbol{g}_t^{(j)}||^2$. Krum uses $i \to j (i \neq j)$ to select the indexes j of the $K - f - 2$ nearest neighbors of $\Delta \boldsymbol{g}_t^{(i)}$, measured by Euclidean distances, where f is the number of malicious clients selected for the aggregation in the system. After computing all the scores of all the weight updates, the FL server uses the weight update with the smallest score to do the aggregation. Meanwhile, other weight updates are dropped.

Coordinate-wise Median (Coomed) [30]. In Coomed, the FL server picks the medians of each coordinate from all the weight updates to build the global weights. Given a set of weight updates $\{\Delta \boldsymbol{g}_t^{(i)}\}_{i=1}^K$ at a communication round t, the FL server uses the Coomed to do the aggregation, which is $\Delta \boldsymbol{g}_t^{Coomed} = Coomed\{\{\Delta \boldsymbol{g}_t^{(i)}\}_{i=1}^K\}$ with $\Delta \boldsymbol{g}_t^{Coomed}$ being a vector with its j^{th} coordinate $\Delta \boldsymbol{g}_t^{Coomed}(j) = med\{\{\Delta \boldsymbol{g}_t^{(i)}(j)\}_{i=1}^K\}$.

FLTrust [54]. In FLTrust, the server collects a root dataset and uses this root dataset to train an auxiliary server model in each communication round. Then, when receiving weight update $\boldsymbol{g}_t^{(i)}$ from a client i, the server calculates a trust score TS_i of the client i, where $TS_i = ReLU(c_i)$ and c_i

1. In this paper, gradients and weight updates are used interchangeably.

is the cosine similarity between $\boldsymbol{g}_t^{(i)}$ and the update of the server model. After calculating all the trust scores, the server updates the global model $\boldsymbol{g}_t = \frac{1}{TS_1+\cdots+TS_K}(TS_1 \cdot \overline{\boldsymbol{g}}_t^{(1)} + \cdots + TS_K \cdot \overline{\boldsymbol{g}}_t^{(K)})$, where $\overline{\boldsymbol{g}}_t^{(i)}$ is the normalized weight updates of the client i.

2.4 Differential Privacy

Current studies show that gradients can reveal the information of the data easily if without proper protection. For example, inference attacks [16] can infer the properties of the data, and deep leakage [17] can reconstruct the data only using the gradients. In FL, if the FL server is somehow compromised or the communication channels between the FL server and clients are monitored, attackers could steal the data information of the clients using such techniques. Thus, from the perspective of clients, preventing the privacy leakage of the gradients becomes a vital problem, though the FL server is regarded as benign in most cases.

A widely used method to protect data privacy is differential privacy [55], which sets up a standard to measure the privacy guarantee of a randomized algorithm.

Definition 1. *A randomized mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ with domain $\mathcal{D}$ and range $\mathcal{R}$ satisfies (ε, δ)-differential privacy if for any two adjacent datasets $\mathsf{d} \in \mathcal{D}$ and $\mathsf{d}' \in \mathcal{D}$ (differ by one sample), and for any subset of mechanism outputs $\mathsf{r} \in \mathcal{R}$, it holds that*

$$Pr[\mathcal{M}(\mathsf{d}) \in \mathsf{r}] \leq e^{\varepsilon} \cdot Pr[\mathcal{M}(\mathsf{d}') \in \mathsf{r}] + \delta, \tag{3}$$

where ε is the privacy budget that measures the privacy level and utility of the randomized algorithm $\mathcal{M}$, and δ indicates the possibility of how likely the privacy is broken.

In FL, since the system consists of the server and several clients, differential privacy can be added on the server or clients [44]. Such difference divides either the methods that implement DP in FL into mainly two categories—local DP (LDP) and central DP (CDP). LDP methods usually add noises to the gradients during the training [42, 47] or add noises to the model updates directly [41, 43], while CDP methods [10, 46] tend to perturb the aggregation functions on the server.

3 Problem Statement

3.1 Threat Model

We adopt a more unrestricted threat model than those applied in previous defenses [1, 19, 22, 54]. The attacker compromises several malicious clients, having complete control over them. For example, the attacker has full access to the local data, and full knowledge related to the local training, including all the training hyper-parameters and model settings. To ensure that the server has valid information for the aggregation, there at least exists one benign client in the system. As assumed in previous approaches, the attacker cannot compromise the FL server. This is because once the attacker can directly manipulate the FL server, the FL training can be easily destroyed due to the dominance capability of the FL server, and the FL training would be meaningless. The attacker can only influence the server indirectly by uploading malicious weight updates using compromised clients. It does not know the data on other benign clients or the aggregation rules on the server. While in order to implement the inference attacks, the attacker may obtain the client updates by eavesdropping on the communication channels between clients and the FL server. Additionally, the attacker can flexibly adjust the attack strategy in order to improve the attacking success rate and camouflage itself. For instance, the attacker can deploy various attacks on different malicious clients and it may not enable the attacks in every communication round.

3.2 Defense Setting

In previous approaches, the defense is considered to be deployed either on the FL server [29, 30, 54] or on the clients [38, 39] by default. While in SIREN$^+$, we perform the defense on both the FL server and clients. Due to the restriction of FL, the server only knows the global model and the weight updates that the clients upload, while it does not have access to the client's data. Besides, in SIREN$^+$, the server also has the capability to collect and keep a small root test dataset that the attacker cannot poison (similar to FLTrust [54]). On each client, compared with existing defenses, SIREN$^+$ enables an additional alarming process to help each client check the integrity of the global model. This alarming process runs locally on each client, guaranteeing that no knowledge of clients will be leaked to the server through this process. This alarming process only uses the local test data, the global models distributed by the server, and the local models derived by the client in each communication round. It only transmits the alarm status to the server through the secure tunnel, without any violation to FL.

4 Methodology

4.1 Overview of SIREN$^+$

Considering that the attacker can only poison the global model via uploading malicious updates [56, 57], existing defenses [32, 58, 59] always implement the detection on the server end. Relying on the single server node makes the FL system vulnerable. Besides, the strict weight analysis used by current Byzantine-robust aggregation rules makes it hard to enable differential private training at the same time. In this case, existing methods can hardly defend against both Byzantine attacks and inference attacks.

To overcome these shortcomings, we propose a new proactive attack-agnostic and differentially private defense system for FL, named SIREN$^+$. Fig. 1 presents the structure of SIREN$^+$ that consists of the design on the FL server end and client end. SIREN$^+$ implements two processes on the client end—the training process and the alarming process. SIREN$^+$ preserves a small partition of the local dataset as local test data on each client. When not enabling LDP, the local training process in SIREN$^+$ is the same as that in the standard FL, responsible for the local training using local data shard. To prevent the client's privacy leakage that may be caused by inference attacks, we deploy LDP on SIREN$^+$ client, injecting appropriate noise during the local training according to the privacy demand of each participant. The alarming process is responsible for testing the global weights. In each communication round, the alarm process on each client checks the global weights by using

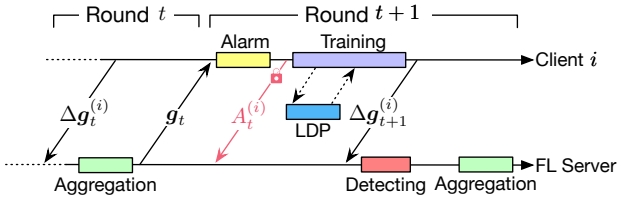


Fig. 2: Workflow of SIREN$^+$ in one communication round.

local weights and the local test dataset. If a client regards the global weight as a poisoned weight, it will alarm the FL server, and the FL server will start the detecting process to exclude the malicious weight updates according to the alarm status of each client. To fully utilize the alarm sent from the clients, we design a decision process to let the FL server detect malicious clients via analyzing both the alarms and weight updates from clients. This client-triggered decision process helps the FL server maintain better model precision and computational efficiency.

4.2 SIREN$^+$ Client's Workflow

Fig. 2 presents SIREN$^+$'s client-end workflow that a client executes an alarming process to verify whether the global model $\boldsymbol{g}_t$ is poisoned, and uploads an alarm status $A_t^{(i)}$ and a model weight update $\Delta \boldsymbol{g}_t^{(i)}$ to the FL server in each communication round. SIREN$^+$ requires each client to keep a local test dataset that could be derived directly from the local training data and a copy of the local model weights generated in the previous round.

The alarming process compares the accuracy between the local model and the global model over the local test dataset to justify whether the global model is trustworthy so that the client can use it for the next round of local training. For simplicity, we use the client i to represent a general participating client, which could be either malicious or benign. Algorithm 1 presents the pseudo-code of the SIREN$^+$ client-end alarming and training processes. A detailed description of the client-end procedure is as follows:

Step 1 (Line 10): When the (t+1)-th communication round begins, the client i receives the global model weight $\boldsymbol{g}_t$ aggregated by the FL server in the previous (*i.e.*, the t-th) communication round.

Step 2 (Line 2-3,11): The client i launches the alarming process (**`Alarm()`**) to evaluate the global model $\boldsymbol{g}_t$ with the help of the local model $\boldsymbol{g}_t^{(i)}$ trained in the previous communication round. The global model $\boldsymbol{g}_t$'s accuracy is ω_t, and the local model $\boldsymbol{g}_t^{(i)}$'s accuracy is $\omega_t^{(i)}$.

Step 3 (Line 4-7,12-15): To justify whether the global model $\boldsymbol{g}_t$ is poisoned, the alarm process further compares the accuracy ω_t and $\omega_t^{(i)}$. If the global model $\boldsymbol{g}_t$ is more accurate than the local model $\boldsymbol{g}_t^{(i)}$, *i.e.*, $\omega_t \geq \omega_t^{(i)} \cdot (1-C_c)$, where C_c is a pre-defined positive threshold, the client i initializes the local model $\boldsymbol{g}^{(i)}$ with $\boldsymbol{g}_t$ in the (t+1)-th communication round training (Line 12-13). Besides, the client i sets the alarm status $A_t^{(i)}$ as 0 (Line 4-5). In contrast, if $\omega_t < \omega_t^{(i)} \cdot (1-C_c)$, then client i initializes the local model $\boldsymbol{g}^{(i)}$ with $\boldsymbol{g}_t^{(i)}$ instead of $\boldsymbol{g}_t$, due to the global model's abnormal performance (Line 14-15). Correspondingly, the client i sets the alarm status $A_t^{(i)}$ to 1 (Line 6-7).

Step 4 (Line 8): The client i sends the alarm status $A_t^{(i)}$ to the FL server in a secure tunnel (*e.g.*, an IPsec tunnel based on the Diffie-Hellman algorithm [60]), which prevents the alarm status from being tampered in network transmission, even when the client i is malicious and generates a false alarm.

The client i obtains a new model $\boldsymbol{g}_{t+1}^{(i)}$ by training the model $\boldsymbol{g}^{(i)}$ on its local data with either a normal optimizer or an LDP optimizer (Sec. 4.3), where the alarm status $A_t^{(i)}$ determines $\boldsymbol{g}^{(i)}$ to be either $\boldsymbol{g}_t^{(i)}$ or $\boldsymbol{g}_t$ in **Step 3**. Then, the client i calculates and sends the local weight update $\Delta \boldsymbol{g}_{t+1}^{(i)} = \boldsymbol{g}_{t+1}^{(i)} - \boldsymbol{g}_t$ to the FL server and stores the local model $\boldsymbol{g}_{t+1}^{(i)}$ for the next round.

This client-end alarming mechanism guarantees that *a poisoned global model always triggers benign clients to alarm*. It should also be noted that malicious clients can deliberately

Algorithm 1: Alarming and Training on SIREN$^+$ clients.

```
// alarming process
1  function Alarm(g_t, g_t^(i))
2      ω_t ← testing g_t on the local test dataset D_0^(i);
3      ω_t^(i) ← testing g_t^(i) on the local test dataset D_0^(i);
4      if ω_t ≥ ω_t^(i) · (1 − C_c) then
           // the global model is normal
5          A_t^(i) ← 0;
6      else
           // the global model is abnormal
7          A_t^(i) ← 1;
8      send A_t^(i) in a secure tunnel to the FL server;
9      return A_t^(i);
   // training process
10 function ClientUpdate(i, g_t)
11     A_t^(i) ← Alarm(g_t, g_t^(i));
12     if A_t^(i) is 0 then
13         g^(i) ← g_t;
14     else
15         g^(i) ← g_t^(i);
16     for each epoch e = 1, ··· , E do
17         if not use DP then
               // normal local training
18             train the model g^(i) with optimizer on the
                local dataset D^(i), and obtain g_{t+1}^(i);
19         else
               // local training with DP
20             train the model g^(i) with LDP on the local
                dataset D^(i), and obtain g_{t+1}^(i);
       // calculate model updates
21     Δg_{t+1}^(i) ← g_{t+1}^(i) − g_t;
22     return Δg_{t+1}^(i);
```

send fake alarms to delude the FL server even when the model is not poisoned. SIREN$^+$ recognizes such delusive alarms from malicious clients at the FL server end. This additional defense makes clients isolated promptly to save the training results of the without-attack rounds once the global model is poisoned, granting SIREN$^+$ a unique feature that can restore the training process even though the server has already been compromised.

4.3 Local Differential Privacy on SIREN$^+$ Clients

Since FL systems may face various types of attacks, the organizer/user may expect to deploy multiple defense methods simultaneously in order to achieve a higher security level. In SIREN$^+$, we enable the LDP technique to provide a convincing defense against inference attacks that could also occur according to the threat model.

The LDP training process implemented in SIREN$^+$ contains mainly four steps. For an arbitrary client i, for each local batch $B_j^{(i)} \in D^{(i)}$, it firstly calculates the gradient $\partial \boldsymbol{g}^{(i)}$ using the local loss function f and model weight $\boldsymbol{g}^{(i)}$:

$$\partial \boldsymbol{g}^{(i)} = \nabla f(B_j^{(i)}, \boldsymbol{g}^{(i)}). \tag{4}$$

Secondly, the gradient $\partial \boldsymbol{g}^{(i)}$ is clipped by the clipping norm $\boldsymbol{C}^{(i)}$:

$$\partial \bar{\boldsymbol{g}}^{(i)} = \partial \boldsymbol{g}^{(i)} / max(1, \frac{||\partial \boldsymbol{g}^{(i)}||_2}{\boldsymbol{C}^{(i)}}). \tag{5}$$

Thirdly, client i injects the Gaussian noise to the clipped gradient $\partial \bar{\boldsymbol{g}}^{(i)}$ with the noise multiplier $\boldsymbol{\sigma}^{(i)}$ and the clipping norm $\boldsymbol{C}^{(i)}$:

$$\partial \tilde{\boldsymbol{g}}^{(i)} = \partial \bar{\boldsymbol{g}}^{(i)} + \mathcal{N}(0, {\boldsymbol{\sigma}^{(i)}}^2 {\boldsymbol{C}^{(i)}}^2 \mathbf{I}). \tag{6}$$

At last, client i updates $\boldsymbol{g}^{(i)}$ with $\partial \tilde{\boldsymbol{g}}^{(i)}$. After the above local training process over the whole local data $D^{(i)}$, client i obtains $\boldsymbol{g}_{t+1}^{(i)}$ (suppose in communication round t) which is then uploaded to the FL server.

4.4 SIREN$^+$ FL Server's Workflow

The FL server trusts neither local model updates nor alarms from any participating clients due to the inherent vulnerability of FL. Before aggregating local model updates and updating the global model as the standard FL does, SIREN$^+$'s FL server first launches a detecting process that analyzes the alarm statuses and evaluates local model weights to identify potential attacks.

In a communication round t, the FL server performs a *two-phase* detection: 1) Examining whether the global model generated in the previous round aggregation (*i.e.*, $\boldsymbol{g}_t$) is poisoned. 2) Testing whether the client model updates collected in the current round (*i.e.*, $\{\Delta \boldsymbol{g}_{t+1}^{(i)} | i \in K\}$) are poisoned. Algorithm 2 presents the pseudo-code of the detecting (**`ServerDetect()`**) and aggregating (**`Aggregation()`**) processes at the FL server. The weight analysis (**`WeightCheck()`**) used in Algorithm 2 is introduced in Sec. 4.6. The following steps illustrate the two-phase detection process of the FL server in each communication round:

Step 1 (Line 18-20): In the t-th communication round, the FL server retrieves alarm status $A_t^{(i)}$ from all participating clients through secure tunnels and collects client model weight updates $\Delta \boldsymbol{g}_{t+1}^{(i)}$, where $i \in K$.

Step 2 (Line 1-16): The FL server analyzes all client alarms following the decision process illustrated in Fig. 3 and Sec. 4.5. If there are no client alarms, the FL server would directly aggregate model weight updates from clients and updates the global model without **Step 3**. However, if there is any alarm, the FL server would further evaluate the model updates $\{\Delta \boldsymbol{g}_{t+1}^{(i)} | i \in S_a\}$ from the clients with activated alarms $A_t^{(i)} = 1$, where $S_a \subseteq K$ and S_a is the set of alarming clients.

Step 3 (Line 21-22): The FL server filters out the client model updates identified as poisonous when aggregating model weight updates to update the global model $\boldsymbol{g}_{t-1}$ from the (t-1)-th communication round, rather than $\boldsymbol{g}_t$, since

Algorithm 2: Detection and aggregation on the SIREN$^+$ FL server.

```
1  function ServerDetect(param)
2      t, K, {A_t^(i)}, {Δg_{t+1}^(i)}, g_t, g_{t-1} ← param;
3      if ∀i ∈ K, A_t^(i) = 0 then
           // no alarms: Case ①②
4          g_{t+1} ← g_t + Σ_{i∈K} α^(i) Δg_{t+1}^(i);
5          return g_{t+1};
6      if ∀i ∈ S_a, max{ω_t^(i) | i ∈ S_a} · (1 − C_s) < ω_t^(i)
         and WeightCheck(ω_t^(i)) then
           // similar accuracies: Case ③
7          if max{ω_t^(i) | i ∈ S_a} · (1 − C_s) ≤
             max{ω_t^(i) | i ∈ S_s} then
               // false alarms
8              S_b ← detect and add benign silent clients;
9              g_{t+1} ← g_t + Σ_{i∈S_b} α^(i) Δg_{t+1}^(i);
10         else
11             S_b ← all the alarming clients;
12             g_{t+1} ← g_{t-1} + Σ_{i∈S_b} α^(i) Δg_{t+1}^(i);
13     else
           // divergent accuracies: Case ④
14         S_b ← detect and add benign alarming clients;
15         g_{t+1} ← g_{t-1} + Σ_{i∈S_b} α^(i) Δg_{t+1}^(i);
16     return g_{t+1};
17 function Aggregation(t, K, g_t, g_{t-1})
18     for each client i ∈ K in parallel do
19         A_t^(i) ← sent back by Alarm(g_t, g_t^(i));
20         Δg_{t+1}^(i) ← ClientUpdate(i, g_t);
21     param ← {t, K, {A_t^(i)}, {Δg_{t+1}^(i)}, g_t, g_{t-1}};
22     g_{t+1} ← ServerDetect(param);
23     return g_{t+1};
24 function FLTraining(T, K, g)
25     g_0, g_1 ← model initialization;
26     for t = 1, 2, ··· , T do
27         g_{t+1} ← Aggregation(t, K, g_t, g_{t-1});
28     return g_{T+1};
```

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3362534

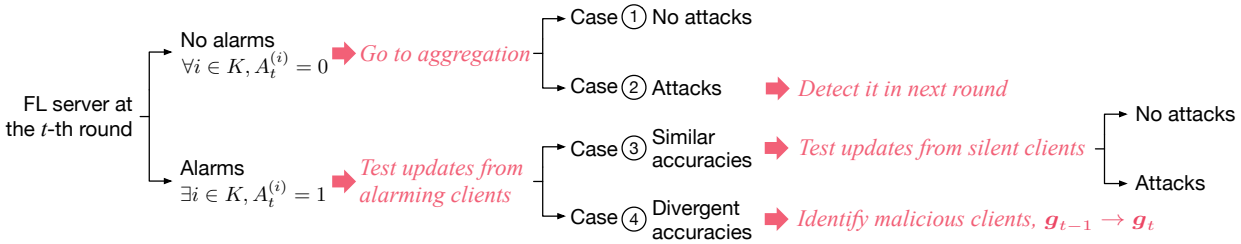


Fig. 3: The FL server's decision process. The decisions made by the FL server are highlighted in red and *italic*.

which is identified as poisoned. Therefore, the global model is updated as $\boldsymbol{g}_{t+1} = \boldsymbol{g}_{t-1} + \sum_{i \in S_b} \alpha^{(i)} \Delta \boldsymbol{g}_{t+1}^{(i)}$, where S_b is the set of clients identified as benign. The FL server also maintains a black list (Sec. 4.7) to exclude the clients identified as malicious for certain times from participating in training.

4.5 Decision Process and Security Analysis

We further analyze the detailed decisions made by the FL server and present the corresponding reasoning. If the FL server receives zero activated alarms at t-th round, there are two possible cases as shown in Fig. 3: Case ① (Line 3-5) $\boldsymbol{g}_t$ is not poisoned, and $\{\Delta \boldsymbol{g}_{t+1}^{(i)} | i \in K\}$ are all benign updates. Case ② (Line 3-5) $\boldsymbol{g}_{t-1}$ is not poisoned, but there are poisoned model updates in $\{\Delta \boldsymbol{g}_{t+1}^{(i)} | i \in K\}$. If $\boldsymbol{g}_t$ is poisoned, the client-end alarming mechanism can guarantee to activate alarms as long as one benign client exists. Besides, Case ② only happens when malicious clients poison the global model $\boldsymbol{g}_1$ at the first attacked communication round. Benign clients will detect such poisoned updates by comparing accuracy of the global model and the local model in the upcoming communication round (Sec. 4.2 Step 3). Thus, the FL server chooses to directly aggregate model updates when there is no alarm.

However, when there are activated alarms, the FL server first tests the accuracy of the models from the clients with activated alarms (*i.e.*, $i \in S_a$) and looks for the maximum accuracy $\max\{\omega_t^{(i)} | i \in S_a\}$ among these clients. We use a user-defined threshold C_s to measure the difference between the maximum accuracy and each alarming client's accuracy. The alarming clients either share a similar accuracy as the Case ③ (Line 6-12) that $\forall i \in S_a$, $\max\{\omega_t^{(i)} | i \in S_a\} \cdot (1 - C_s) < \omega_t^{(i)}$, or have divergent accuracies that $\exists i \in S_a$, $\max\{\omega_t^{(i)} | i \in S_a\} \cdot (1 - C_s) \geq \omega_t^{(i)}$ as the Case ④ (Line 13-15).

For Case ③, if there is no attack—neither the global model $\boldsymbol{g}_{t-1}$ nor client updates $\{\Delta \boldsymbol{g}_t^{(i)} | i \in K\}$ are not poisoned, the activated alarms must be *false alarms* deliberately generated by malicious clients. If there are attacks and the alarming clients' model updates have similar accuracies, then we should test the silent clients' model updates to further verify whether the alarming clients' model updates are all poisoned or all benign. Thus, for Case ③ we should always test all the silent clients' model updates $\{\Delta \boldsymbol{g}_t^{(i)} | i \in S_s\}$, where S_s is the set of silent clients. If the silent clients' highest accuracy is close to or even better than the alarming clients' highest accuracy:

$$\max\{\omega_t^{(i)} | i \in S_a\} \cdot (1 - C_s) \leq \max\{\omega_t^{(i)} | i \in S_s\}, \quad (7)$$

the FL server can assure that the benign clients are silent, and thus, all alarming clients' updates are poisoned. If the accuracy of a silent client's updates is close to the maximum accuracy of all silent clients, we believe this client is benign. So we add a silent client to the benign client set S_b, when its accuracy matches $\max\{\omega_t^{(i)} | i \in S_s\} \cdot (1 - C_s) < \omega_t^{(i)}$, where $i \in S_s$. Since all benign clients are silent, the alarms are generated by malicious clients as *false alarms*, and the global model from the last round $\boldsymbol{g}_t$ is not poisoned.

Contrarily, for Case ③, if the maximum accuracy of silent clients is lower than the maximum accuracy of the alarming clients: $\max\{\omega_t^{(i)} | i \in S_a\} \cdot (1 - C_s) > \max\{\omega_t^{(i)} | i \in S_s\}$, then all silent client's model updates are poisoned, and all alarming clients are benign due to their similar accuracies.

For Case ④, the divergent accuracies of the alarming clients indicate that both benign and malicious clients are alarming. Again, we use the maximum accuracy of all alarming clients to filter out the alarming malicious clients. If an alarming client's accuracy satisfies $\max\{\omega_t^{(i)} | i \in S_a\} \cdot (1 - C_s) < \omega_t^{(i)}$, then we add it to the benign client set S_b. Since benign clients always alarm when detecting a poisonous $\boldsymbol{g}_t$, there are no benign clients among the silent clients. Thus, we ignore all silent clients in this case.

Besides, we jointly apply accuracy checking and weight analysis to recognize malicious clients and achieve better detection in Sec. 4.6. To analyze the security level of SIREN$^+$, we can transfer the conclusion of FLTrust. Since there exists at least one client in the SIREN$^+$'s FL system is benign, we can regard such client's model update as the server model update in FLTrust. Both SIREN$^+$ and FLTrust use cosine similarity as the criterion, and SIREN$^+$ also uses accuracy, which is more direct and strict, as the additional criterion. Thus, the security level of SIREN$^+$ is no less than FLTrust.

4.6 Weight Analysis

Since most of the attacks aim to generate malicious weight updates, which can have reverse impacts on the global model compared with benign weight updates, these malicious weight updates usually represent reverse changing directions of the model compared with benign updates. With this intuition, similar to FLTrust [54], weight analysis is added into SIREN$^+$ (`WeightCheck()`). However, SIREN$^+$ only uses the information from clients while executing the weight analysis. Such attribute is essentially different from FLTrust, which has a predefined expectation of the global model via using an auxiliary model trained by the data on the server to evaluate client weight updates. With weight analysis in SIREN$^+$, the server not only compares the accuracy between the update with max accuracy and

other updates but also compares the angles between these updates. If the angle between an update ω_i and the update with max accuracy $\max\{\omega_t^{(i)}|i \in S_a\}$ is greater than $\pi/2$, then ω_i would be regarded as a malicious update by the server. Otherwise, ω_i is regarded as a benign update and can be calculated into the global model. Via weight analysis, the server can check the updates from clients through another perspective while keeping its objectivity.

4.7 Penalty and Award Mechanisms

To further improve the capability of SIREN$^+$, some auxiliary mechanisms are also applied to SIREN$^+$'s architecture that is introduced in Sec. 4.2 and Sec. 4.4. All these auxiliary mechanisms are used only by the server so that clients do not have any extra computational burdens. The server can flexibly determine whether to use these auxiliary mechanisms according to the computational resources on the server as well as the demand for better security and performance.

Penalty Mechanism: We design a penalty mechanism to improve the stability of SIREN$^+$. Since malicious clients can attack the server consistently and the corresponding consistent checks waste a huge amount of computational resources. With the penalty mechanism, the server records the number of instances in which each client is regarded as malicious. If a client's count exceeds a threshold C_p, the server will no longer accept updates from this client without checks, considering it malicious by default. With this method, the server can effectively save the computational overheads and improve the stability of the system.

Award Mechanism: The penalty mechanism may misjudge benign clients to be malicious clients because of the variance of the data on each client. Thus, the server exploits an award mechanism to rejoin a banned client into the training with a probability. In a communication round, if a banned client is regarded as a benign client by the server, the penalty count of this client would reduce C_a, according to the award mechanism. If the penalty count of this banned client is less than C_p, then this client could participate in the training process again. With this mechanism, the server can alleviate the side effects of the penalty mechanism.

5 EVALUATION

We implement an FL prototype of SIREN$^+$ based on TensorFlow [61] and use the `multiprocessing` library to launch multiple processes to simulate multiple clients in the system. We implement the LDP part using *TensorFlow Privacy* [62], a Python library including several customized TensorFlow optimizers/models that could train neural networks with differential privacy. We use the *DPAdamOptimizer* with *QuantileAdaptiveClipSumQuery* to dynamically adjust the hyper-parameters $\boldsymbol{C}^{(i)}$ during the local training. Their design patterns follow DP-SGD [63] and adaptive clipping [64]. We have open-sourced SIREN$^+$ at GitHub[2].

For Byzantine attacks, we evaluate SIREN$^+$ with four distinctive attacking methods: sign-flipping attack, label-flipping attack, adaptive attack, and targeted model poisoning attack, and compare SIREN$^+$ with three prevailing Byzantine-robust aggregation methods—multi-Krum, Coomed, and FLTrust. **For inference attacks**, we evaluate SIREN$^+$ with two types of MIAs: threshold MIA and LR MIA. Our evaluation experiments run on an NVIDIA Tesla V100 GPU and two NVIDIA 2080Ti GPUs.

2. https://github.com/AISIGSJTU/Siren-Plus

TABLE 2: Default settings of main parameters.

	Description	IID	Non-IID
B	Local batch-size	64	64
T	Communication rounds	40	40
E	Local training epochs	5	5
p	Non-IID degree	0	0.5
C_c	Client identification threshold	4%	4%
C_s	Server identification threshold	10%	10%
C_p	Penalty mechanism threshold	$0.45 \cdot T$	$0.45 \cdot T$
C_a	Award mechanism parameter	0.5	0.5

TABLE 3: The architecture of the CNN model used for Fashion-MNIST under the non-DP setting.

Layer Type	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$5 \times 5 \times 64$
Convolution + ReLU + Dropout	$5 \times 5 \times 64$
Fully Connected + ReLU + Dropout	128
Softmax	10

5.1 Experimental Setup

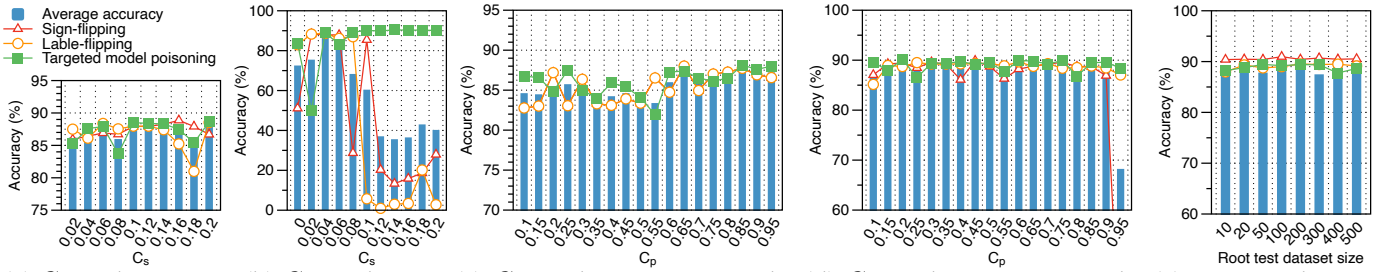
Table 2 summarizes the main parameters of SIREN$^+$ for the image classification task on the Fashion-MNIST dataset. For the CIFAR-10 dataset, we reuse most of the parameters while changing C_s to 15% (for both IID and non-IID data) and changing C_c to 6% (only for non-IID data). We also extend T to 100 for the CIFAR-10 dataset to train the ResNet-18.

Datasets: We evaluate SIREN$^+$ on two mainstream computer vision datasets: Fashion-MNIST and CIFAR10. Fashion-MNIST consists of 60K training gray-scale images and 10K testing gray-scale images from 10 classes. CIFAR-10 contains 50K training color images and 10K testing color images.

Models: We train SIREN$^+$ using two types of CNN models and ResNet-18 [65] on Fashion-MNIST and CIFAR-10 datasets, respectively. For the Fashion-MNIST dataset, we use a computationally complex CNN model for the non-DP setting (shown in Table 3) and a simple CNN model for DP setting (shown in Table 4). For the CIFAR-10 dataset, we use ResNet-18 to verify the effectiveness of SIREN$^+$ using a larger model.

IID and Non-IID Data: For IID training data, we randomly split the whole training data into $|K|$ shards and allocate these data shards to $|K|$ clients directly. For non-IID training data, we follow the non-IID pattern in the paper of adaptive attack [19] to generate non-IID data for clients using the non-IID degree p. By using p, training data with label l is distributed into lth group of clients with possibility p. In this case, a higher p indicates a higher degree of non-IID. We set $p = 0.5$ in all the experiments over non-IID training data.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3362534



(a) C_s on the server. (b) C_c on clients. (c) C_p on the FL server with only the penalty mechanism. (d) C_p on the FL server with both the penalty and award mechanisms. (e) Root dataset size on the FL server.

Fig. 4: Searching the configuration of important hyper-parameters.

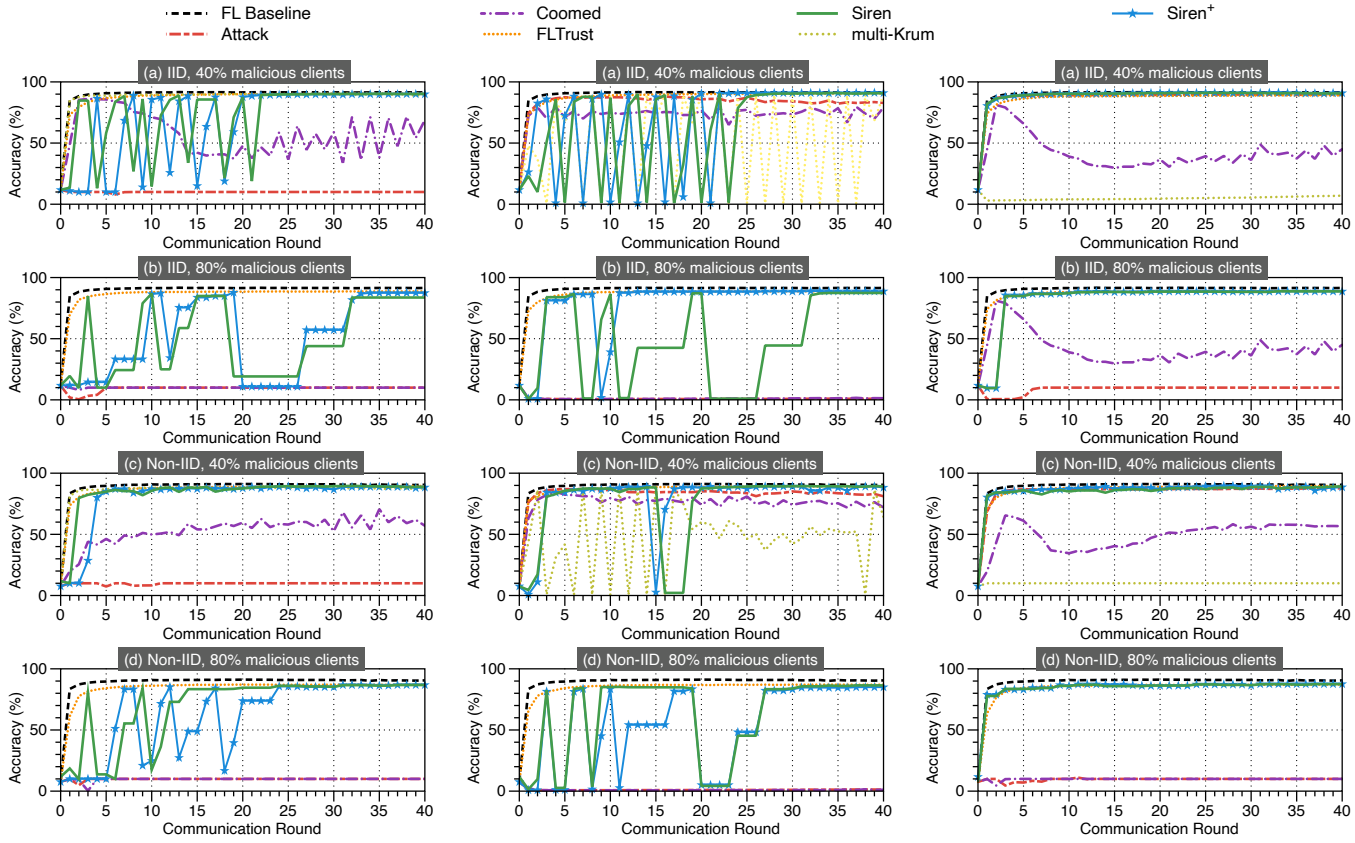


Fig. 5: Training efficiency under sign-flipping attack when $|K| = 10$.

Fig. 6: Training efficiency under label-flipping attack when $|K| = 10$.

Fig. 7: Training efficiency under adaptive attack when $|K| = 10$.

TABLE 4: The architecture of the CNN model used for Fashion-MNIST under DP setting.

Layer Type	Size
Input	$28 \times 28 \times 1$
Convolution + ReLU	$8 \times 8 \times 16$
Max Pooling	2×1
Convolution + ReLU	$4 \times 4 \times 32$
Max Pooling	2×1
Fully Connected + ReLU	32
Softmax	10

Evaluation Metrics: We compare SIREN$^+$ with several prevailing Byzantine-robust aggregation methods mainly from model performance, system robustness, and training efficiency. To compare the model performance, we use accuracy as the metric. To compare the system's robustness, we test each method under various malicious settings to observe the trend of the training process. To compare the training efficiency, we use the convergence speed as the criteria. We also use the AUC of the inference attacker model under SIREN$^+$ and original SIREN to demonstrate the effectiveness of SIREN$^+$. Besides, we define a metric—*malicious index*—to indicate how frequently a client is recognized to be malicious by the server in SIREN$^+$, representing the server's recognition accuracy to malicious clients.

The root test dataset on the server: SIREN$^+$ uses a root test dataset on the server to recognize potentially malicious clients. We randomly pick out 100 samples (Sec. 5.2) from the training dataset, then distribute the remaining data to each client, as the server's root test dataset should be collected by the server itself instead of being derived from clients. This small root test dataset shares the same data

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3362534

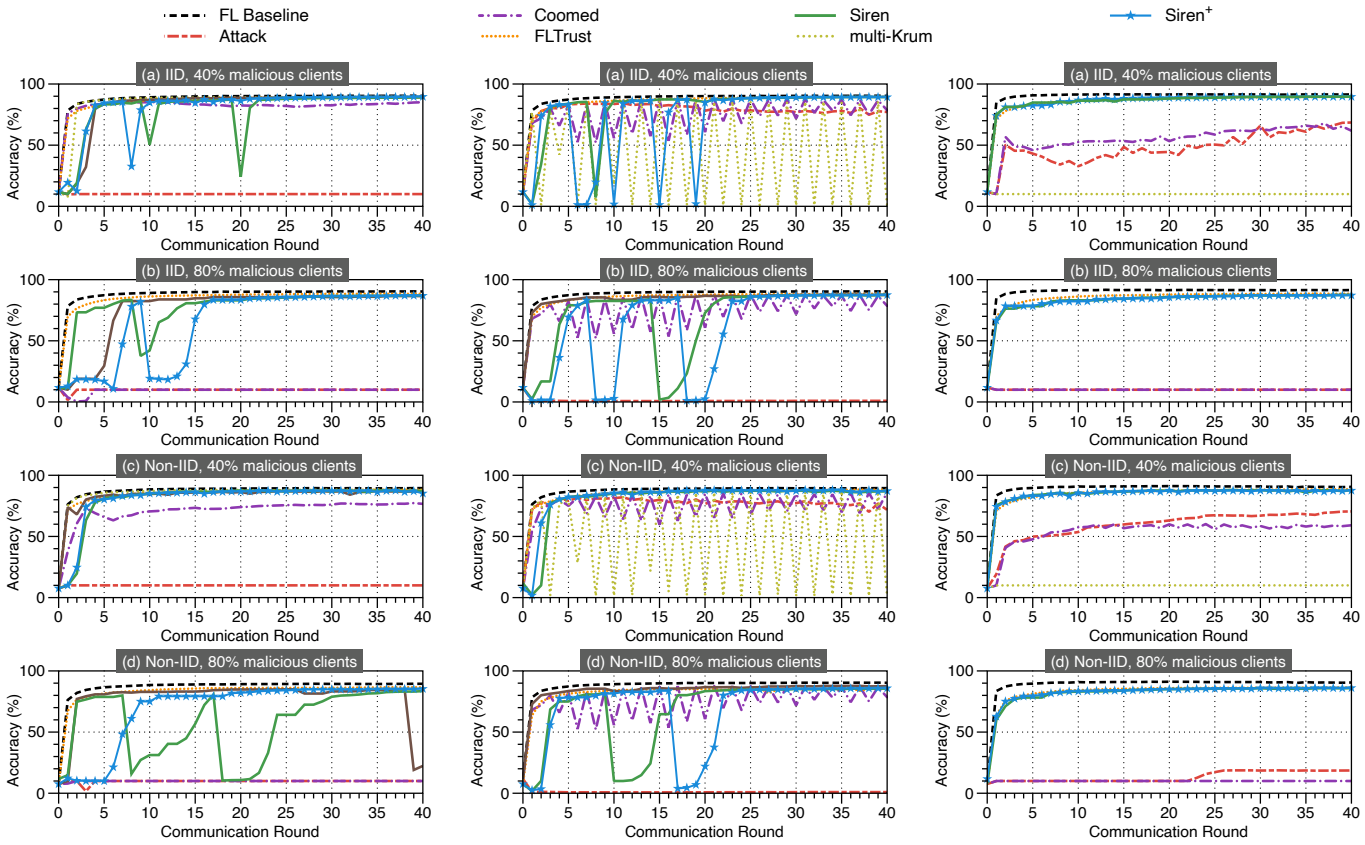


Fig. 8: Training efficiency under sign-flipping attack when $|K| = 50$.

Fig. 9: Training efficiency under label-flipping attack when $|K| = 50$.

Fig. 10: Training efficiency under adaptive attack when $|K| = 50$.

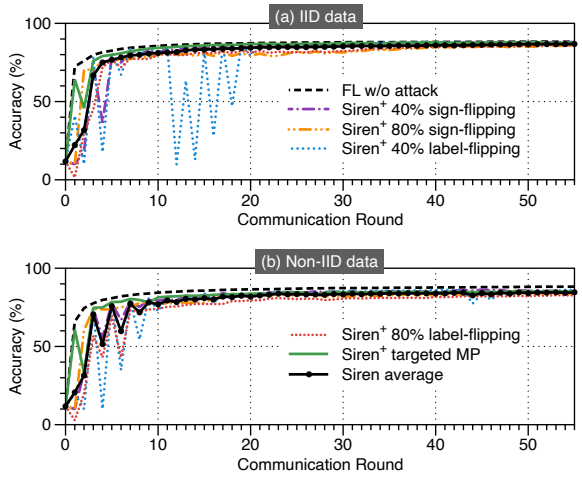


Fig. 11: Training efficiency of SIREN$^+$ when $|K| = 200$.

distribution with the data on each client when using the IID data setting, and the root test dataset shares a different data distribution with the data on each client when using the non-IID data setting. Our experiments test both scenarios.

5.2 Parameter Selection

Since SIREN$^+$ has more user-customized parameters compared with other prevailing Byzantine-robust aggregation methods, we implement an ablation study of SIREN$^+$'s optimal main parameters for both the FL server and clients. The exploration results are presented in Figure 4.

Figure 4(b) and Figure 4(a) show the impacts of C_s and C_c on the global model's accuracy, respectively. For both C_c and C_s, a larger value represents a higher tolerance for the various data distribution on clients, while such tolerance can offer attackers more operable space. Besides, SIREN$^+$ is non-sensitive to C_s while sensitive to C_c because clients trigger the detecting process in SIREN$^+$.

Figure 4(c) and Figure 4(d) show the exploration results of C_p with penalty mechanism only, and with both penalty mechanism and award mechanism, respectively. When only using the penalty mechanism, the accuracy fluctuates with the increase of C_p. However, SIREN$^+$ is more independent from C_p when using both penalty and award mechanism, representing that the award mechanism can make the penalty mechanism more stable and complete. With both penalty and award mechanisms, various penalty schemes could be designed for different purposes.

Figure 4(e) shows the sensitivity of the root dataset size in SIREN$^+$. Compared with FLTrust, SIREN$^+$ only uses the server's data for testing. Even when the size of the root test dataset equals 10, SIREN$^+$ is still effective. Our experiments choose 100 as the root test dataset size, with which the global model achieves the highest accuracy.

5.3 Defending against Untargeted Byzantine Attacks

This section evaluates SIREN$^+$ with multi-Krum, Coomed, and FLTrust under three types of prevailing untargeted Byzantine attacks, *i.e.*, sign-flipping attack, label-flipping attack, and adaptive Krum attack. To demonstrate the robustness of SIREN$^+$, we set the proportion of malicious clients to 40% (typical condition) and 80% (extreme condition)

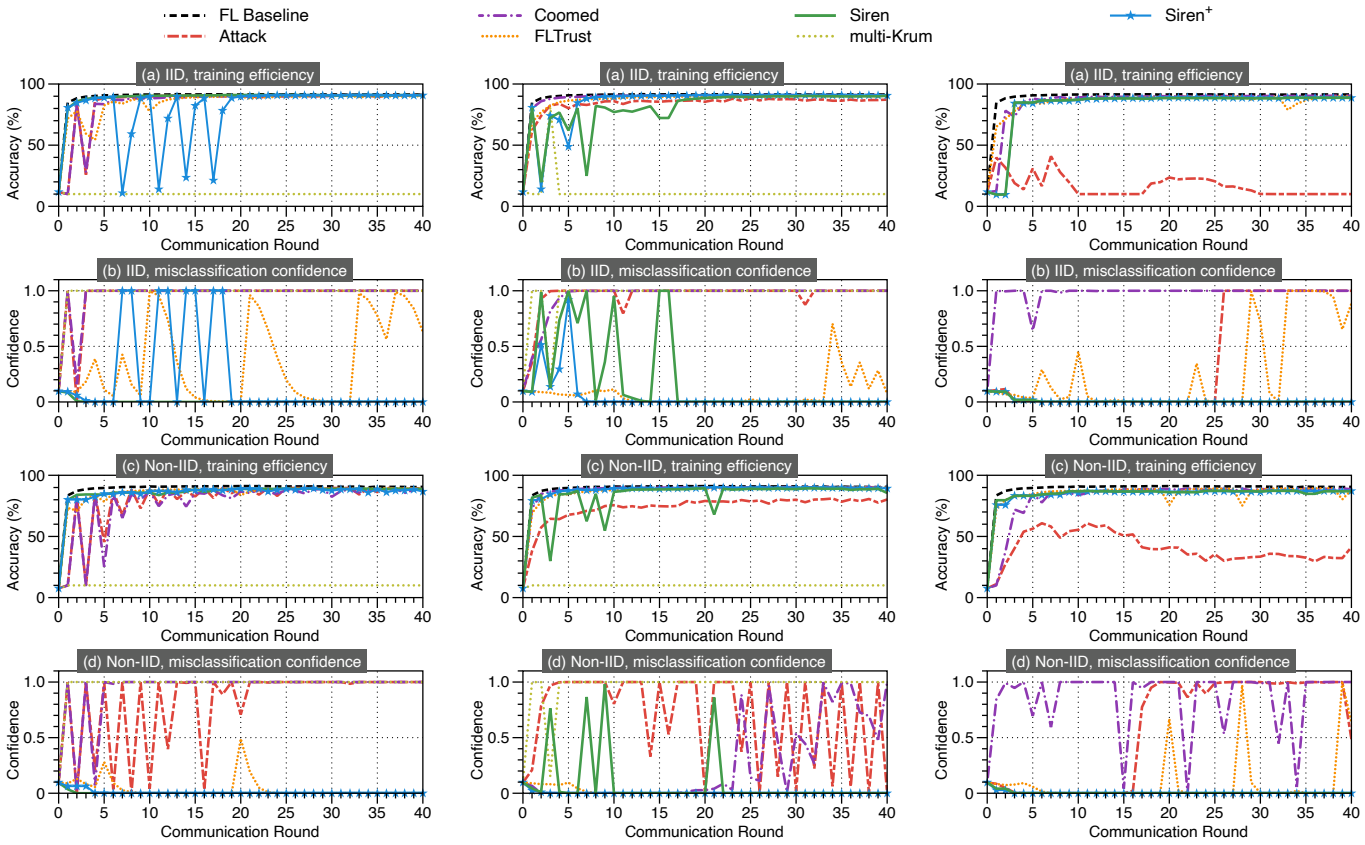


Fig. 12: Training efficiency and misclassification confidence under targeted model poisoning, $|K| = 10$.

Fig. 13: Training efficiency under simultaneously multiple attacks (10% for each attack, 40% total) attack when $|K| = 10$.

Fig. 14: Training efficiency under simultaneously multiple attacks (20% for each attack, 80% total) attack when $|K| = 10$.

over IID and non-IID data, respectively. To demonstrate the scalability of SIREN$^+$, we conduct experiments in the 10-client system, the 50-client system, and the 200-client system.

10-client system: The training processes of SIREN$^+$ and other three types of Byzantine-robust aggregation methods under sign-flipping attack are illustrated in Fig. 5. When the malicious client proportion is 40%, both multi-Krum, FLTrust, and SIREN$^+$ successfully defend against the attack over IID and non-IID data, while Coomed fails, shown in Fig. 5(a) and Fig. 5(c). SIREN$^+$ outperforms multi-Krum, while SIREN$^+$ and FLTrust tie, both reaching a near-baseline performance. When the malicious client proportion is 80%, both multi-Krum (cannot be initialized properly) and Coomed fail to protect the system, while SIREN$^+$ and FLTrust can still defend against the attack, shown in Fig. 5(b) and Fig. 5(d). Due to the high malicious client proportion in the system that only the updates from two benign clients can be used during the aggregation, the global models trained by SIREN$^+$ and FLTrust both have noticeable accuracy drops, which are more than 3%. However, such reductions in the accuracy are quite small and acceptable considering the extreme condition.

Fig. 6 shows the performance of SIREN$^+$, multi-Krum, Coomed, and FLTrust under label-flipping attack with both 40% and 80% malicious client proportion over IID and non-IID data, respectively. The training process under label-flipping attack is similar to the training process under sign-flipping attack, while multi-Krum can not even defend against label-flipping attack when 40% of the clients in the system are malicious over both IID and non-IID data at this time. Only SIREN$^+$ and FLTrust can always offer reliable protection, within 2% accuracy drops over IID data and 4% accuracy drops over non-IID data.

Fig. 7 shows the performance of SIREN$^+$, multi-Krum, Coomed, and FLTrust under adaptive attack with both 40% and 80% malicious client proportion over IID and non-IID data individually. Only SIREN$^+$ and FLTrust can protect the system as before. Multi-Krum and Coomed both obtain very poor model accuracy, which is less than 50%, while the baseline accuracy is more than 90%. The accuracy decrease of the global model trained by SIREN$^+$ is smaller than 2% over IID data and 3% over non-IID data, which outperforms FLTrust in most cases.

50-client system: Fig. 8, Fig. 9, and Fig. 10 show the training processes of SIREN$^+$, multi-Krum, Coomed, and FLTrust under sign-flipping attack, label-flipping attack, and adaptive attack, respectively. Due to the increasing number of benign clients in the system, the multi-Krum and Coomed methods obtain more accurate models in the 50-client system than in the 10-client one. However, since the same increasing number of malicious clients in the 50-client system, the training process trends of multi-Krum and Coomed are similar to their trends in the 10-client system. Both multi-Krum and Coomed experience severe accuracy fluctuations, such as more than 20% accuracy degradation. SIREN$^+$ and

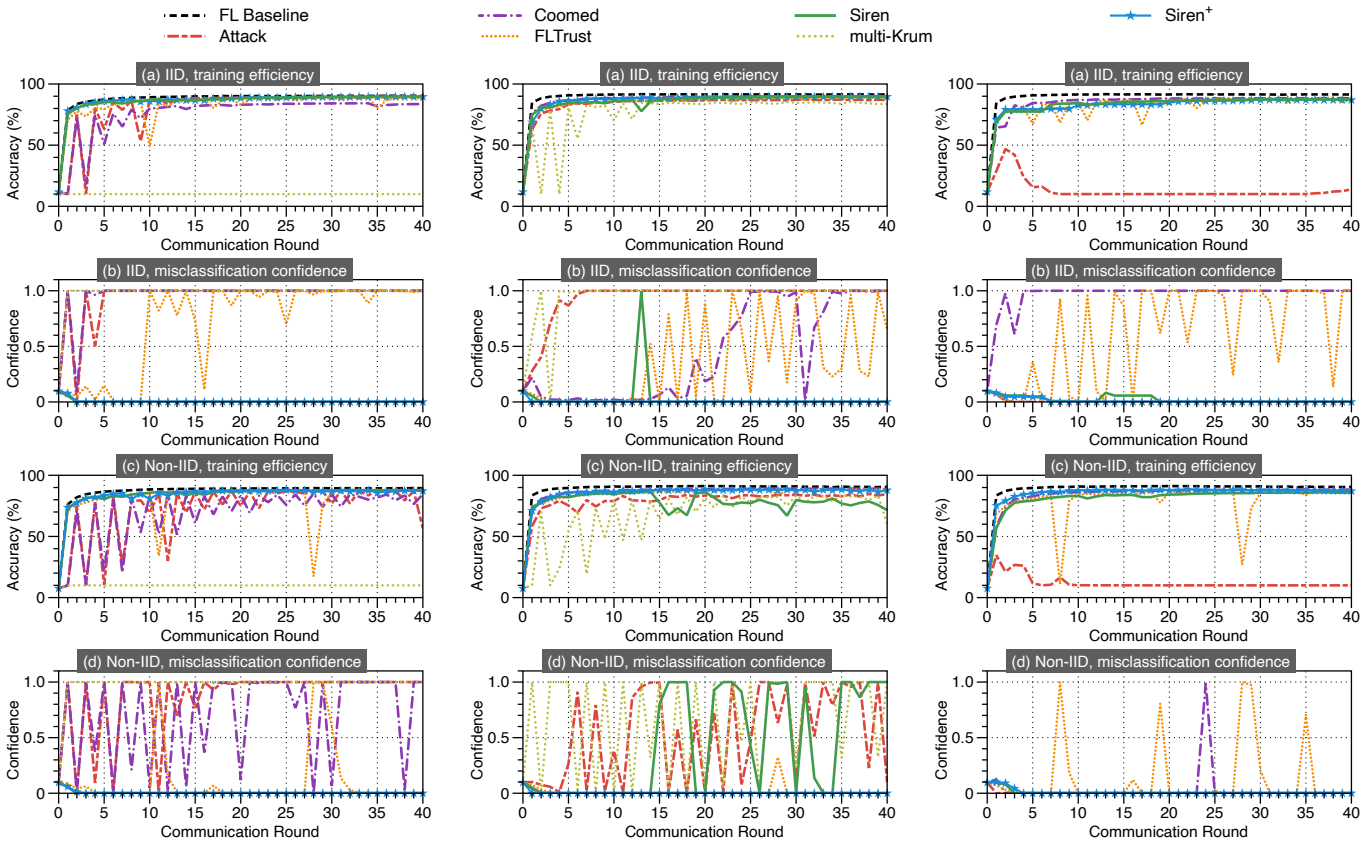


Fig. 15: Training efficiency and misclassification confidence under targeted model poisoning, $|K| = 50$.

Fig. 16: Training efficiency under simultaneously multiple attacks (10% for each attack, 40% total) attack when $|K| = 50$.

Fig. 17: Training efficiency under simultaneously multiple attacks (20% for each attack, 80% total) attack when $|K| = 50$.

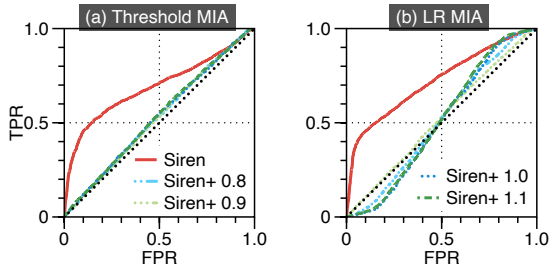


Fig. 18: ROC curve when SIREN$^+$ under MIA with and without LDP at the final round (the black dot line indicates a random classifier).

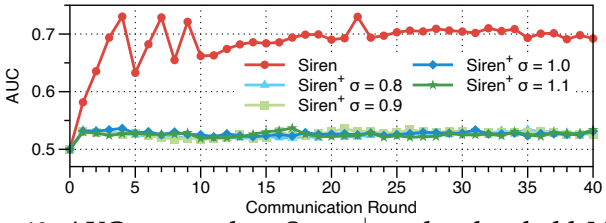


Fig. 19: AUC curve when SIREN$^+$ under threshold MIA with and without LDP.

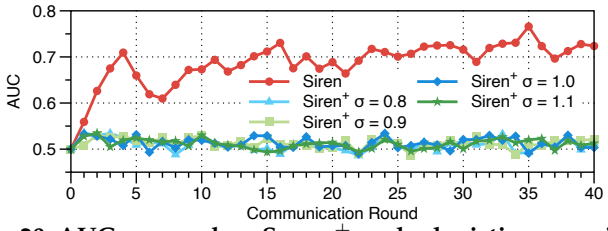


Fig. 20: AUC curve when SIREN$^+$ under logistic regression MIA with and without LDP.

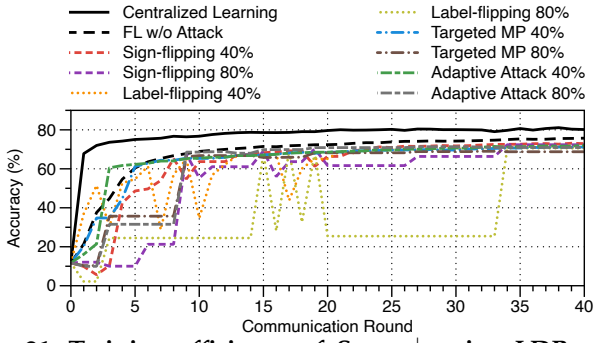


Fig. 21: Training efficiency of SIREN$^+$ using LDP when $|K| = 10$.

FLTrust tie in the 50-client system setting, both of which can still defend against all the three types of untargeted attacks while achieving more than 85% accuracy in most cases.

200-client system: Fig. 11(a) shows the training curves of SIREN$^+$ and SIREN in the 200-client system under sign-flipping attack, label-flipping attack with both 40% and 80% malicious clients over IID data. Fig. 11(b) shows the training curves over non-IID data with the same settings. Both SIREN$^+$ and SIREN successfully defend against the untargeted Byzantine attacks in both typical and extreme conditions while achieving near-baseline performance. The

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3362534

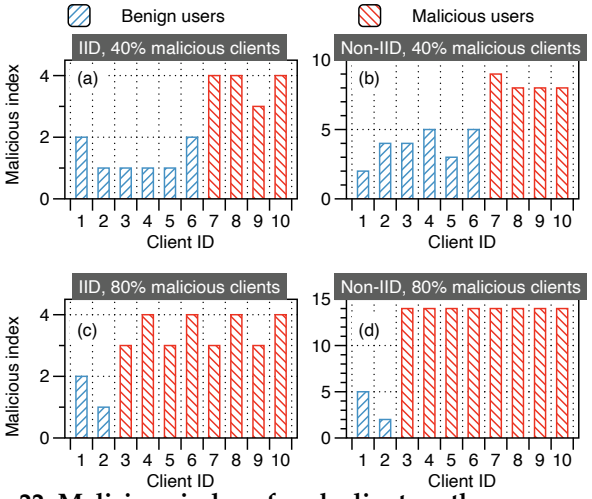


Fig. 22: Malicious index of each client on the server under sign-flipping attack when $|K| = 10$ using SIREN.

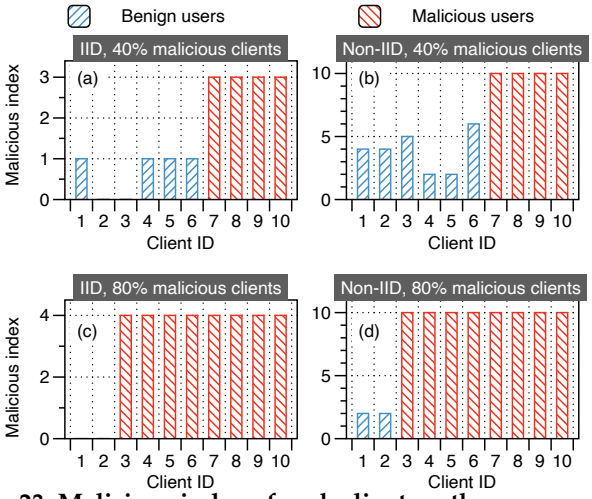


Fig. 23: Malicious index of each client on the server under label-flipping attack when $|K| = 10$ using SIREN.

accuracy drops of SIREN$^+$ over IID and non-IID data are less than 3% and 5% in most cases, respectively. The overall performance of SIREN$^+$ is similar to SIREN's average performance, while SIREN$^+$ reaches higher model accuracy in some cases. This is similar to the performance of SIREN$^+$ in the 10-client system and 50-client system, indicating that SIREN$^+$ is capable of providing effective protection against untargeted Byzantine attacks from small-scale systems to large-scale systems.

5.4 Defending against Targeted Byzantine Attack

This section evaluates SIREN$^+$ along with multi-Krum, Coomed, and FLTrust under targeted model poisoning attack. Compared with the previously mentioned untargeted Byzantine attacks, defending against targeted model poisoning attack is a more challenging task since such attack only twists specific predictions. Besides the training curves, we also use the misclassification confidence [22] to illustrate the attacking effect of a targeted Byzantine attack.

10-client system: Fig. 12 shows the training curves and misclassification confidence of SIREN$^+$ with other defensive methods under the targeted model poisoning attack from 40% malicious clients. The training curves in Fig. 12(a)

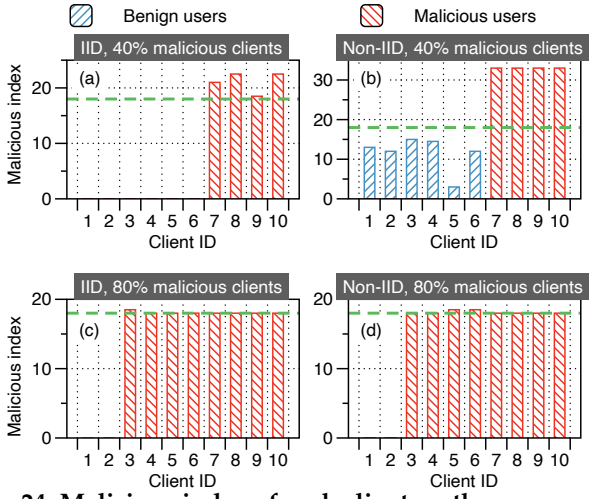


Fig. 24: Malicious index of each client on the server under sign-flipping attack when $|K| = 10$ using SIREN$^+$ (The green dash line represents C_p).

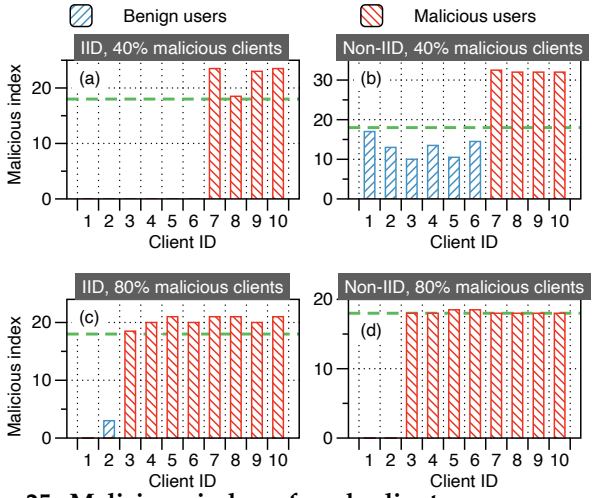
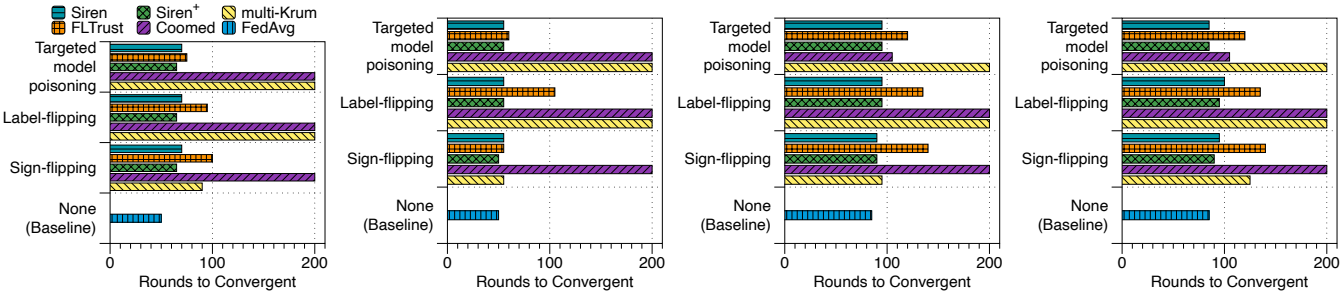


Fig. 25: Malicious index of each client on server under label-flipping attack when $|K| = 10$ using SIREN$^+$ (The green dash line represents C_p).

and Fig. 12(c) show that Coomed, SIREN$^+$, and FLTrust all perform well, but Krum fails, which is aligned to the original paper of targeted model poisoning attack [22]. Since the targeted model poisoning attack attacks a global model while keeping it performing normally, we continue to analyze the misclassification confidence of the global models trained by three effective Byzantine-robust aggregation methods. Fig. 12(b) and Fig. 12(d) show that only SIREN$^+$ can defend against targeted model poisoning attacks since SIREN$^+$ successfully keeps the global model's misclassification confidence at a low level during the whole training process. Such an observation is more obvious in the non-IID setting. The results in Fig. 12 indicate that only SIREN$^+$ can defend against both untargeted and targeted Byzantine attacks.

50-client system: Fig. 15 illustrates the training curves and misclassification confidence of SIREN$^+$ along with the other three defensive methods in the 50-client system. Compared with the training curves in the 10-client system, the targeted model poisoning attack is more stealthy in the 50-client system. Fig. 12(a) and Fig. 12(c) show that all four defensive



(a) $|K| = 10$, IID data. (b) $|K| = 10$, Non-IID data. (c) $|K| = 50$, IID data. (d) $|K| = 50$, Non-IID data.

Fig. 26: Communication rounds needed for being convergent under different scenarios. If the rounds reach 200, it means that the global model is attacked successfully or cannot achieve an accuracy of 85%.

methods' training process trends in the 50-client system are similar to the 10-client system. Fig. 12(b) and Fig. 12(d) show that only SIREN$^+$ can still defend against the targeted Byzantine attack with a constant and low misclassification confidence.

200-client system: Fig. 11 shows the training curves of SIREN$^+$ under targeted model poisoning attack over both IID data and non-IID data in the 200-client system. SIREN$^+$ can successfully defend against targeted model poisoning attacks and achieve relatively high accuracy—less than 2% lower than the baseline.

5.5 Defending against Inference Attacks

We first evaluate SIREN$^+$ under two MIAs—threshold MIA and logistic regression (LR) MIA—over the CIFAR-10 dataset. Threshold MIA is an untrained MIA, while LR MIA is a trained MIA. Fig. 18 presents the ROC curve of the attacker models at the final communication round. The black dot line indicates a random binary classifier, and the red line represents the original SIREN without LDP. The other lines record SIREN$^+$ using LDP with different privacy levels. With LDP, SIREN$^+$ can substantially weaken the discrimination of the attacker model of both two MIAs and reduce the AUC of the attacker models from ∼70% to ∼50%, which approximates the random classifier's performance. The effectiveness of SIREN$^+$'s defense against MIAs persists during the whole training process, as shown in Fig. 19 and Fig. 20.

Secondly, we test SIREN$^+$ under all the targeted Byzantine attacks and untargeted Byzantine attacks used in the previous sections with LDP to further ensure that the LDP part does not contradict SIREN$^+$'s defense against Byzantine attacks. Since introducing local differential privacy may cause accuracy drops on benign clients and local updates may be more divergent, we reuse the main parameters over non-IID data in this experiment. This experiment tests SIREN$^+$ over IID data since non-IID data leads to constant fluctuations in the training process. Besides, to simulate a more real-world scenario, we only enable LDP on benign clients and disable it on malicious clients since attackers are assumed to fully control the whole training process. Fig. 21 presents SIREN$^+$'s training curves under various Byzantine attacks in the 10-client system, where SIREN$^+$'s accuracy drops are less than 3% in most of the cases, indicating that LDP does not interfere SIREN$^+$'s defense against Byzantine attacks.

5.6 The Auxiliary Mechanisms' Effectiveness

Fig. 22 and 23 visualize the malicious index of each client under sign-flipping attack and label-flipping attack with the basic SIREN, respectively. When the malicious client proportion equals 40%, the malicious index of a benign client is close to the malicious index of a malicious client. Nearly half of the benign clients are incorrectly recognized to be malicious, indicating that only using the penalty mechanism in the basic SIREN can cause apparent false positives. Thus, we introduce the award mechanism and an adaptive C_p to alleviate such side effects of the penalty mechanism in SIREN$^+$. Fig. 24 and Fig. 25 illustrate that the award mechanism and an adaptive C_p can jointly magnify the difference of the malicious index between benign and malicious clients to reduce the false positives.

5.7 Compatibility and Generality Analysis

Compatibility: To demonstrate the compatibility of SIREN$^+$, we train a more complicated model—ResNet-18—on the CIFAR-10 dataset. Table 5 shows that SIREN$^+$ can achieve a higher model accuracy under three types of untargeted Byzantine attacks at different proportions of malicious clients. Compared with the simple CNN model trained on the Fashion-MNIST dataset, the accuracy drops are amplified due to the increasing number of malicious clients. When the malicious client proportion is 20%, SIREN$^+$ can defend against all three attacks while achieving ideal model accuracy, with less than 4% accuracy drops. When the proportion reaches 40%, the model accuracy of SIREN$^+$ slightly drops less than 5.5% accuracy loss. However, when the proportion reaches 80%, the accuracy drops violently. We argue that it is because larger models are more sensitive to a decrease in the volume of benign training data, resulted from the increase of malicious clients.

Generality: Since attackers may implement various Byzantine attacks at the same time in real-world scenarios, we also test SIREN$^+$ under multiple attacks simultaneously to demonstrate its generality. Fig. 13 and Fig. 14 present the training curves of SIREN$^+$ with other three types of defensive methods and the misclassification confidence of targeted model poisoning under 40% multiple Byzantine attacks (10% for each) and 80% multiple attacks (20% for each) in the 10-client system respectively. Only SIREN$^+$ is general enough to defend against all the three types of untargeted Byzantine attacks and the targeted Byzantine attack simultaneously, while the other defensive methods

TABLE 5: Model accuracy over CIFAR-10 using IID data distribution with ResNet-18 when $|K| = 20$.

Attack Type	Malicious Client Proportion	Accuracy
None	0%	80.23%
Sign-flipping	20%	76.32%
	40%	74.4%
	80%	63.81%
Label-flipping	20%	76.94%
	40%	75.42%
	80%	64.53%
Adaptive Attack	20%	78.41%
	40%	75.79%
	80%	61.57%

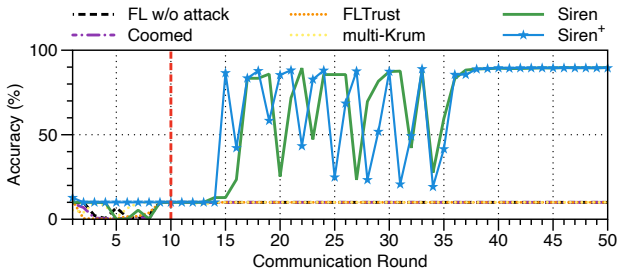


Fig. 27: Training efficiency under sign-flipping attack when $|K| = 10$, 40% malicious clients and the defense delay is 10 rounds.

fail. Fig. 16 and Fig. 17 illustrate that SIREN$^+$ is also effective in even larger-scale systems.

5.8 Computational Efficiency Analysis

Compared with FEDAVG, SIREN$^+$ causes extra computational overheads on each client due to the local testing in the alarming process. Since the local test dataset is quite small compared with the local training dataset, the alarming process does not burden clients too much.

From the space complexity perspective, SIREN$^+$ needs extra storage on both the FL server and clients to store the consecutive models. However, compared to the size of training datasets, the storage space for extra models is fairly small.

We also compare the convergence speed of SIREN$^+$ with the three defensive methods. Fig. 26 presents that the extra communication rounds taken by SIREN$^+$ are negligible, compared with multi-Krum, Coomed, and FLTrust. Compared with existing defensive methods, SIREN$^+$ can provide near-baseline convergent speed while maintaining more robust training.

5.9 Recovering Compromised Training

We design an experiment to evaluate how SIREN$^+$ can recover FL when the server is compromised. In the first ten rounds, we prevent the server from using any defensive methods. After the 10^{th} round, the server is recovered and begins to use defensive methods. As Fig. 27 shows, only SIREN$^+$ can recover the training process, and finally obtain a global model performing similarly to the model trained under no attacks.

6 Related Work

FL was originally motivated by data privacy concerns. However, due to its distributed nature, FL is rather vulnerable and can be attacked by different kinds of attacks. Many studies on FL's attacking and defense methodologies emerged in recent years.

Attacks: Byzantine attacks and inference attacks are two mainstream attack categories in FL. Byzantine attacks aim to poison the global model by sending malicious model updates to the FL server through compromised clients. According to the attacking scope, existing Byzantine attacks can be divided into two categories: untargeted Byzantine attacks and targeted Byzantine attacks. Most existing attacking methodologies focus on degrading the whole performance of the global model for testing dataset [18, 19, 25, 58], referred to as *untargeted attack*. Another type of Byzantine attack is known as *targeted attack* [14, 22, 23, 24, 66], seeking to poison the global model on some specific data samples (or labels) in a targeted manner without degrading the global model's performance for the rest data. Though the effect of Byzantine attacks is always related to the number of compromised clients [49], this genre of attack is the most common attack in FL. While the goal of inference attacks is to derive the data characteristics of clients. It can be divided into mainly four categories, which are *Class Representative Inference Attack* [26], *Membership Inference Attack* [16, 27, 52], *Property Inference Attack* [16], and *Input&Label Inference Attack* [17, 28]. Among them, the most widely used inference attack is *Membership Inference Attack*.

Defenses: Byzantine-robust FL methods mainly focus on alleviating the negative effects of Byzantine attacks. Since the median of local model weights is a robust estimator to defend against Byzantine attacks, many studies have adopted and improved this strategy [29, 30, 31, 32, 58, 59]. However, existing Byzantine-robust methods suffer from a few practical issues, such as false negatives triggered by non-IID data and vulnerabilities to a large proportion of malicious clients. Zeno [67] and Zeno++ [68] effectively address such weaknesses with a stochastic first-order oracle that grades each client's update and only aggregates updates with high scores, which also falls into the category of weight analysis-based defense methodology. FLTrust [54] calculates a trust score for each client model update based on the cosine similarity between the server's model updates and clients' updates. Then, the FL server uses the trust score as the weight of clients' updates when updating the global model. Sear [36] utilizes encryption to enhance the security of communication between the FL server and clients. DPBFL [37] improves the privacy and robustness of FL via shuffled aggregation. While Flip [38] and Fl-wbc [39] attempt to alleviate the data poisoning attacks through the client's efforts. Unlike existing defensive methods, SIREN$^+$ jointly exploits accuracy checking and weight analysis. Particularly, SIREN$^+$ crafts a proactive alarming mechanism that orchestrates all participating clients and the FL server to effectively detect attacks.

Differential Privacy: FL with differential privacy mainly focuses on how to add noise to gradients to protect the data privacy of clients. Some studies use differential privacy technique only on the client side [40, 41, 42] or the server

side [10, 46], while other recent studies mainly add such noise on both the FL-server side and client side [43, 44]. In this paper, since LDP is enough for SIREN$^+$ to defend against two types of MIAs, we do not inject such noise into the server's aggregation.

7 Discussion

Compared with existing robust aggregation rules in FL, our SIREN$^+$ is the first method that concentrates on the collaboration between the FL server and clients. With this feature, SIREN$^+$ can easily defend against attacks from a large portion of malicious clients, even when malicious clients simultaneously conduct different types of attacks. SIREN$^+$'s robustness against such extreme scenarios is a giant leap compared with previous FL defenses. Future attackers may design more sophisticated attacks corrupting fewer parameters in the model update and fewer training rounds by detecting sensitive parameters and training rounds, and apply collaborative attack strategies. Existing defenses, including SIREN$^+$, may be vulnerable under such attacks. Similarly, more fine-grained and efficient defenses are needed to handle more stealthy attacks. The modularized defense that can flexibly deal with various genres of attacks based on different system environments is also a promising direction for efficient and comprehensive FL defenses.

8 Conclusion

This paper proposes SIREN$^+$, a comprehensively robust defense system for federated learning that can effectively defend against both Byzantine attacks and inference attacks. Instead of using model weight analysis that is incompatible with DP algorithms, SIREN$^+$ jointly utilizes accuracy checking and LDP to defend against both Byzantine attacks and inference attacks. SIREN$^+$'s accuracy checking is based on a distributed alarming mechanism that defends FL from all types of Byzantine attacks in real-world scenarios, such as, up to 80% malicious client proportion. Besides, SIREN$^+$ is resilient to both trained and untrained MIAs by carefully injecting noise to local updates at client ends with LDP. Extensive experiments with different Byzantine attack methods on IID and non-IID data prove the effectiveness of SIREN$^+$, compared with other state-of-the-art defense methods, such as multi-Krum, coordinate-wise median, and FLTrust. The experiments under two types of MIAs additionally illustrate that SIREN$^+$ is capable of nullifying the inference attacker model at the same time.

References

[1] H. Guo, H. Wang, T. Song, Y. Hua, Z. Lv, X. Jin, Z. Xue, R. Ma, and H. Guan, "Siren: Byzantine-robust Federated Learning via Proactive Alarming," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2021.

[2] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, vol. 10, no. 3152676, pp. 10–5555, 2017.

[3] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh *et al.*, "Federated Learning: Strategies for Improving Communication Efficiency," in *NeurIPS Workshop on Private Multi-Party Machine Learning (NeurIPS Workshop)*, 2016.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient Learning of Deep Networks from Decentralized Data," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[5] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.

[6] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed *et al.*, "Scaling Distributed Machine Learning with the Parameter Server," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[7] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei *et al.*, "Petuum: A New Platform for Distributed Machine Learning on Big Data," *IEEE Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

[8] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[9] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated Machine Learning: Concept and Applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, 2019.

[10] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning Differentially Private Recurrent Language Models," in *International Conference on Learning Representations (ICLR)*, 2018.

[11] N. Rieke, J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein *et al.*, "The future of digital health with federated learning," *NPJ digital medicine*, vol. 3, no. 1, pp. 1–7, 2020.

[12] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, "Federated learning for healthcare informatics," *Journal of Healthcare Informatics Research*, vol. 5, no. 1, pp. 1–19, 2021.

[13] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: vision, hype and reality for data privacy and protection," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[14] L. Lyu, H. Yu, and Q. Yang, "Threats to Federated Learning: A Survey," *arXiv preprint arXiv:2003.02133*, 2020.

[15] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[16] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting Unintended Feature Leakage in Collaborative Learning," in *IEEE Symposium on Security and Privacy (S&P)*, 2019.

[17] L. Zhu, Z. Liu, and S. Han, "Deep Leakage from Gradients," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[18] 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 *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[19] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local Model Poisoning Attacks to Byzantine-Robust Federated Learning," in *USENIX Security Symposium (USENIX Security)*, 2020.

[20] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen, "Learning to Detect Malicious Clients for Robust Federated Learning," *arXiv preprint arXiv:2002.00211*, 2020.

[21] C. Xie, O. Koyejo, and I. Gupta, "Fall of Empires: Breaking Byzantine-tolerant SGD by Inner Product Manipulation," in *Uncertainty in Artificial Intelligence (UAI)*, 2020.

[22] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. B. Calo, "Analyzing Federated Learning through an Adversarial Lens," in *International Conference on Machine Learning (ICML)*, 2019.

[23] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to Backdoor Federated Learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[24] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "DBA: Distributed Backdoor Attacks against Federated Learning," in *International Conference on Learning Representations (ICLR)*, 2019.

[25] V. Shejwalkar and A. Houmansadr, "Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning," in *Network and Distributed System Security (NDSS) Symposium*, 2021.

[26] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep Models under the GAN: Information Leakage from Collaborative Deep Learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[27] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-Box

Inference Attacks Against Centralized and Federated Learning," in *IEEE Symposium on Security and Privacy (S&P)*, 2019.

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

[29] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[30] D. Yin, Y. Chen, K. Ramchandran, and P. L. Bartlett, "Byzantine-robust Distributed Learning: Towards Optimal Statistical Rates," in *International Conference on Machine Learning (ICML)*, 2018.

[31] X. Cao, J. Jia, and N. Z. Gong, "Provably Secure Federated Learning against Malicious Clients," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[32] Y. Chen, L. Su, and J. Xu, "Distributed Statistical Machine Learning in Adversarial Settings: Byzantine Gradient Descent," in *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 2017.

[33] E. M. El Mhamdi, R. Guerraoui, and S. Rouault, "The Hidden Vulnerability of Distributed Learning in Byzantium," in *International Conference on Machine Learning (ICML)*, 2018.

[34] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "DRACO: Byzantine-resilient Distributed Training via Redundant Gradients," in *International Conference on Machine Learning (ICML)*, 2018.

[35] D. Alistarh, Z. Allen-Zhu, and J. Li, "Byzantine Stochastic gradient descent," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[36] L. Zhao, J. Jiang, B. Feng, Q. Wang, C. Shen, and Q. Li, "Sear: Secure and efficient aggregation for byzantine-robust federated learning," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 19, no. 5, pp. 3329–3342, 2021.

[37] X. Ma, X. Sun, Y. Wu, Z. Liu, X. Chen, and C. Dong, "Differentially private byzantine-robust federated learning," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 33, no. 12, pp. 3690–3701, 2022.

[38] K. Zhang, G. Tao, Q. Xu, S. Cheng, S. An, Y. Liu, S. Feng, G. Shen, P.-Y. Chen, S. Ma *et al.*, "Flip: A provable defense framework for backdoor mitigation in federated learning," *arXiv preprint arXiv:2210.12873*, 2022.

[39] J. Sun, A. Li, L. DiValentin, A. Hassanzadeh, Y. Chen, and H. Li, "Fl-wbc: Enhancing robustness against model poisoning attacks in federated learning from a client perspective," *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[40] S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy, and W. Wei, "LDP-Fed: Federated Learning with Local Differential Privacy," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking (EdgeSys)*, 2020.

[41] M. Seif, R. Tandon, and M. Li, "Wireless Federated Learning with Local Differential Privacy," in *IEEE International Symposium on Information Theory (ISIT)*, 2020.

[42] Y. Zhao, J. Zhao, M. Yang, T. Wang, N. Wang, L. Lyu, D. Niyato, and K.-Y. Lam, "Local Differential Privacy-Based Federated Learning for Internet of Things," *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 8836–8853, 2020.

[43] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, "Federated Learning with Differential Privacy: Algorithms and Performance Analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.

[44] M. Naseri, J. Hayes, and E. De Cristofaro, "Local and Central Differential Privacy for Robustness and Privacy in Federated Learning," *arXiv preprint arXiv:2009.03561*, 2020.

[45] H. B. McMahan, G. Andrew, U. Erlingsson, S. Chien, I. Mironov, N. Papernot, and P. Kairouz, "A General Approach to Adding Differential Privacy to Iterative Training Procedures," *arXiv preprint arXiv:1812.06210*, 2018.

[46] R. C. Geyer, T. Klein, and M. Nabi, "Differentially Private Federated Learning: A Client Level Perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[47] R. Hu, Y. Guo, H. Li, Q. Pei, and Y. Gong, "Personalized Federated Learning with Differential Privacy," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9530–9539, 2020.

[48] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin *et al.*, "Federated Learning with Non-IID Data," *arXiv preprint arXiv:1806.00582*, 2018.

[49] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning," in *IEEE symposium on security and privacy (S&P)*. IEEE, 2022.

[50] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A Novel Image Dataset for Benchmarking Machine Learning Algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[51] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," *Tech Report*, 2009.

[52] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership Inference Attacks Against Machine Learning Models," in *IEEE symposium on security and privacy (S&P)*, 2017.

[53] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models," *arXiv preprint arXiv:1806.01246*, 2018.

[54] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping," *arXiv preprint arXiv:2012.13995*, 2020.

[55] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our Data, Ourselves: Privacy via Distributed Noise Generation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, 2006.

[56] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Model Poisoning Attacks in Federated Learning," in *NeurIPS Workshop on Security in Machine Learning (SecML)*, 2018.

[57] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data Poisoning Attacks Against Federated Learning Systems," *arXiv preprint arXiv:2007.08432*, 2020.

[58] C. Xie, O. Koyejo, and I. Gupta, "Generalized Byzantine-tolerant SGD," *arXiv preprint arXiv:1802.10116*, 2018.

[59] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Defending Against Saddle Point Attack in Byzantine-robust Distributed Learning," in *International Conference on Machine Learning (ICML)*, 2019.

[60] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[61] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis *et al.*, "TensorFlow: A System for Large-Scale Machine Learning," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[62] G. et al, "TensorFlow Privacy," *https://github.com/tensorflow/privacy*, 2018.

[63] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep Learning with Differential Privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (CCS)*, 2016.

[64] G. Andrew, O. Thakkar, B. McMahan, and S. Ramaswamy, "Differentially Private Learning with Adaptive Clipping," *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[65] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

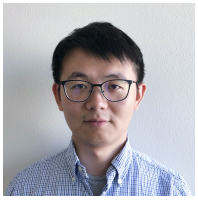
[66] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, "Can You Really Backdoor Federated Learning?" *arXiv preprint arXiv:1911.07963*, 2019.

[67] C. Xie, S. Koyejo, and I. Gupta, "Zeno: Distributed Stochastic Gradient Descent with Suspicion-based Fault-tolerance," in *International Conference on Machine Learning (ICML)*, 2019.

[68] ——, "Zeno++: Robust Fully Asynchronous SGD," in *International Conference on Machine Learning (ICML)*, 2020.

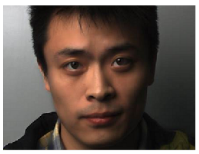
This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2024.3362534

Hanxi Guo is currently a Master's student of Computer Science in the School of Electronic information and Electrical Engineering at Shanghai Jiao Tong University, Shanghai, China, where he also received his Bachelor's degree in 2020. His research interests center around the security and application of federated learning and distributed computing.



Hao Wang is currently an assistant professor in the CSE Division at Louisiana State University, Baton Rouge, specializing in distributed computing and machine learning systems. He received his Ph.D. degree from the University of Toronto and his Bachelor's and Master's degrees from Shanghai Jiao Tong University. He is a recipient of the NSF CRII Award.

Tao Song (Member, IEEE) is currently an assistant professor in the Department of Computer Science and Engineering at Shanghai Jiao Tong University. He received his Ph.D. degree in computer science and M.Eng. degree in software engineering from Shanghai Jiao Tong University. His research interests include distributed machine learning, cloud/distributed computing and system security.



Yang Hua is presently a lecturer at Queens University Belfast, United Kingdom. He received his Ph.D. degree from Université Grenoble Alpes/Inria Grenoble RhoneAlpes, France, funded by Microsoft Research 's Inria Joint Center. He has won four titles of prestigious international competitions in the field of computer vision and machine learning.

Ruhui Ma is currently an associate professor in the Department of Computer Science and Engineering at Shanghai Jiao Tong University. He received his Ph.D. degree in computer science from Shanghai Jiao Tong University. His research interests include cloud computing systems, AI systems, and machine learning.

Xiulang Jin is currently a researcher in Huawei 2012 laboratory. He received the B.S. degree in mathematics and applied mathematics, and the M.E degree in instruments science and technology from the Harbin Institute of Technology in 2017 and 2019, respectively. His current research interests include federated learning, AI model robustness and model encryption protection.

Zhengui Xue is currently a postdoctoral researcher at Queens University Belfast, United Kingdom. She received her Ph.D. degree from the NUS Graduate School for Integrative Sciences and Engineering, National University of Singapore in 2013. She held a postdoctoral position at the Ecole Nationale des Travaux Publics de l'Etat from 2014 to 2015. Her research interests include adaptive systems, knowledge-based intelligent control, optimal control and machine learning.

Haibing Guan (Member, IEEE) is currently a professor in the Department of Computer Science and Engineering at Shanghai Jiao Tong University, and the director of the Shanghai Key Laboratory of Scalable Computing and Systems. He received his Ph.D. degree from Tongji University in 1999. His research interests include computer architecture, cloud/distributed computing, system security and distributed machine learning.