CrowdGuard: Federated Backdoor Detection in Federated Learning

Phillip Rieger*
Technical University of Darmstadt
phillip.rieger@trust.tu-darmstadt.de

Torsten Krauß*
University of Würzburg
torsten.krauss@uni-wuerzburg.de

Markus Miettinen
Technical University of Darmstadt
markus.miettinen@tu-darmstadt.de

Alexandra Dmitrienko
University of Würzburg
alexandra.dmitrienko@uni-wuerzburg.de

Ahmad-Reza Sadeghi
Technical University of Darmstadt
ahmad.sadeghi@trust.tu-darmstadt.de

***Abstract*—Federated Learning (FL) is a promising approach enabling multiple clients to train Deep Neural Networks (DNNs) collaboratively without sharing their local training data. However, FL is susceptible to backdoor (or targeted poisoning) attacks. These attacks are initiated by malicious clients who seek to compromise the learning process by introducing specific behaviors into the learned model that can be triggered by carefully crafted inputs. Existing FL safeguards have various limitations: They are restricted to specific data distributions or reduce the global model accuracy due to excluding benign models or adding noise, are vulnerable to adaptive defense-aware adversaries, or require the server to access local models, allowing data inference attacks.**

This paper presents a novel defense mechanism, CrowdGuard, that effectively mitigates backdoor attacks in FL and overcomes the deficiencies of existing techniques. It leverages clients' feedback on individual models, analyzes the behavior of neurons in hidden layers, and eliminates poisoned models through an iterative pruning scheme. CrowdGuard employs a server-located stacked clustering scheme to enhance its resilience to rogue client feedback. The evaluation results demonstrate that CrowdGuard achieves a 100% True-Positive-Rate and True-Negative-Rate across various scenarios, including IID and non-IID data distributions. Additionally, CrowdGuard withstands adaptive adversaries while preserving the original performance of protected models. To ensure confidentiality, CrowdGuard uses a secure and privacy-preserving architecture leveraging Trusted Execution Environments (TEEs) on both client and server sides.

I. Introduction

Federated Learning (FL) allows multiple clients to collaboratively train a Deep Neural Network (DNN) on their private data. In contrast to centralized learning approaches, in FL each client trains its own DNN locally and shares only the trained parameters of the model with an aggregation server [56]. Thus, FL reduces concerns regarding the privacy of the clients' local data, as they never leave the respective client, which is especially important in times of increased privacy awareness, legal restrictions, and regulations [2], [3], [4]. FL also improves on the resource usage, as the computationally expensive training is parallelized and outsourced to the participating clients. As a result, FL has become a popular technology and is applied in various applications, including image recognition [74], [78], [79], [80], e.g., between multiple hospitals [33], [74], [79], natural language processing (NLP), e.g., text prediction on smartphones [34], [57], personalization [17], risk classification [25], or threat detection in IoT networks [64].

However, outsourcing the training process to individual clients makes FL vulnerable to poisoning attacks. Here, an adversary compromises a subset of the clients and lets them submit manipulated model updates. Such attacks can be untargeted [24], [46], [75] or targeted (so-called backdoor attacks) [7], [66], [81], [86], [92]. In the following, we will focus on targeted poisoning attacks that are more challenging to detect (cf. Sect. III-B). These attacks cause the aggregated model to misbehave at prediction time (also called inference) if the input sample for the DNN contains a specific adversary-controlled trigger. Moreover, in FL, the clients must trust the server because several attacks have successfully inferred information about the training data from the trained parameters of a model [29], [36], [52], [63], [71], [76], [82], [88], [95].

The current defenses can be broadly classified into two categories: Influence Reduction (IR) approaches [6], [7], [15], [58], [62], [94] and Detection and Filtering (DF) approaches [10], [28], [61], [65], [73], [81], [96], [44]. IR techniques aim to limit the impact of poisoned updates on the model, while DF techniques try to detect and remove the poisoned updates. These defenses employ techniques such as clipping, noising, subgroup training, distance metrics, and client-side analysis of the final predictions using the clients' local data. However, the existing defenses still face several challenges. First, they often assume that the training data of different clients are independently and identically distributed (IID). Hence, these defenses may not work effectively when the training data of different clients differ, thus are non-IID. Second, adaptive attackers aware of the defense mechanisms can bypass these defenses. Finally, existing defenses do not address the problem of unauthorized access to local models, allowing inference attacks. We will elaborate on related work

*These authors contributed equally to this work

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23233
www.ndss-symposium.org

in detail in Sect. VII.

Our goals and contributions: We present CrowdGuard, a backdoor-resilient and privacy-enhancing FL architecture that effectively overcomes the limitations of existing solutions. The key rationale of CrowdGuard is to leverage a secure client-feedback-loop, where clients conduct local validation and analyze changes in the behavior of individual neurons. We define a new Hidden Layer Backdoor Inspection Metric (HLBIM) that enables CrowdGuard to identify poisoned models through iterative pruning based on multiple significance tests for analyzing the models' behavior. Although the adversary might be able to inject a backdoor without affecting the final predicted class on regular data, it cannot avoid changing the behavior of at least a subset of deep-layer neurons to introduce the backdoor functionality into the DNN. To mitigate manipulated feedback from malicious clients, the server employs a multi-layer clustering scheme to aggregate the feedback of different validation clients. Utilizing the client-feedback-loop, analyzing the changes in the behavior of individual neurons, and employing stacked robust aggregation of clients' feedback enable CrowdGuard to effectively identify poisoned models without making assumptions about the attack or data scenarios. Using the clients' data provides a comprehensive overview of the clients' training objectives, allowing to identify both, benign and poisoned models, even in scenarios where all clients' data are disjoint (non-IID).

To mitigate the privacy risk and prevent the feedback-loop from allowing malicious clients to perform inference attacks on the received local models, CrowdGuard leverages secure enclaves and remote attestation to prevent unauthorized access to the local models. By extending this concept to the server, we effectively solve the problem of combining privacy-preserving aggregation with backdoor mitigation. Thus, CrowdGuard guarantees that clients' data remains confidential and cannot be inferred from the local models.

Our contributions include:

- We propose CrowdGuard, an architecture that enables secure and privacy-preserving utilization of clients' local data for local model inspection. Thus, we provide the foundation for a new class of poisoning detection algorithms. Additionally, we remove the need for trust in the aggregation server by utilizing secure, attestable enclaves on the server side and combining the backdoor detection algorithm with an efficient, secure aggregation (Sect. IV-A).
- We design a novel backdoor detection algorithm that analyzes the *hidden layer outputs* of local models to distinguish between benign and backdoored models. As the adversary's primary goal is to change the model's predictions for inputs containing the trigger, it cannot disguise the backdoor in all hidden layers without reducing the attack impact. The in-depth analysis and iterative pruning enable CrowdGuard to effectively identify benign and poisoned updates even in non-IID data scenarios against sophisticated, defense-adapted attacks (Sect. IV-B).
- We conducted an extensive evaluation of the efficiency and effectiveness of CrowdGuard on various FL scenarios to analyze different attack parameters, including various non-IID scenarios, poisoning rates, backdoor types, and datasets such as CIFAR-10 [43] and MNIST [21]. By analyzing changes in the behavior of the neurons, CrowdGuard achieved 100% True-Positive-Rates (TPRs) and True-Negative-Rates (TNRs) in a wide range of scenarios, outperforming existing defenses while avoiding their limitations. The runtime evaluation showed an acceptable overhead of 29.5 seconds on average for the client-side validation of 20 models using enclaves on the CPU (Sect. V).

We are currently integrating CrowdGuard's source code into the OpenFL framework [27] to ensure that CrowdGuard can be used not just for further research in this area but also for real-world applications [1].

II. Background

In the following, we describe the necessary background about Federated Learning (FL) in Sect. II-A and targeted poisoning attacks on FL in Sect. II-B.

A. Federated Learning

Federated Learning (FL) [41], [56], [93] is used for generating or improving a shared machine learning (ML) model, i.e., a Deep Neuronal Network (DNN), by collaborative efforts of multiple clients $C_k \in \{C_1, \dots C_N\}$ and a server $\mathcal{S}$ in an iterative process [56]. The major benefit of FL is that the clients C_k use their local data $\mathcal{D}_k$ for the training process. These client-located data do not have to be shared with the server $\mathcal{S}$. Hence, such training is more privacy-preserving than vanilla ML. Additionally, the extensive computations during learning are distributed to multiple clients, so no cost-intensive infrastructure is needed at $\mathcal{S}$.

The learning process takes place over multiple *FL rounds* that are supervised by the server $\mathcal{S}$. At the start of each round t, $\mathcal{S}$ first deploys a global model G^t to a randomly selected group of clients $C_i \in \{C_1, \dots, C_n\}$, which is a subset of the total N clients. The n chosen clients initialize their local model L_i^t with G^t and continue using their local dataset $\mathcal{D}_i$ to train the new local model L_i^t. The training is a regular learning process configured by a number of hyper-parameters like, e.g., the learning rate, that optimizes one loss function. Afterward, $\mathcal{S}$ collects all $L_1^t, \dots, L_n^t$ models and aggregates them to a new global model G^{t+1}, by averaging the differences and adding them to G^t [56].

In detail, $\mathcal{S}$ computes the update of each weight for each model, builds the average of these contributions and adds the resulting value to the global model [56]. This algorithm is called *FederatedAveraging (FedAVG)*. The final contributions are weighted with the global learning rate δ (see Eq. 1) [42]. After computing a new global model G^{t+1}, $\mathcal{S}$ can initialize a new round, e.g., until a round limit is reached.

$$G^{t+1} = G^t + \delta(\frac{1}{n}\sum_{i=1}^{n}(L_i^t - G^t)) \tag{1}$$

Client Data Distributions. One major point that influences the performance of any FL system regarding model accuracy but also complicates the security mechanisms against malicious contributions is the underlying distribution of the training datasets $\mathcal{D}_k \in \{\mathcal{D}_1, \dots, \mathcal{D}_N\}$ [48], [37]. Even if the overall

[1] https://github.com/TRUST-TUDa/crowdguard

(a) Pixel (b) Semantic (c) Benign

Fig. 1: Comparison of Backdoor Triggers.

amount of samples for each label in the whole system over all clients $C_1, \ldots, C_N$ is equal, the data is unlikely to be uniformly distributed among all clients so that all $\mathcal{D}_k$ follow the same distribution [56]. Contrary to an *independent and identically distributed (IID)* data scenario, a *non-IID* case naturally delivers divergent trained models from each client. Non-IID can manifest with different severities [97], i.e., all clients can have samples of all available labels, but the overall count for each label differs [49]. It is common to introduce a peak non-IID rate of $q \in [0, 1]$ in sample counts for one label, the so-called *main label* of the client [14], [65]. The rest of the labels are assumed to follow a uniform distribution [14], [65]. Alternatively, one can consider a specific distribution for all sample counts, like the Dirichlet [59] or normal distribution, with the peak at the main label.

Another situation is when some clients miss one label entirely or only have data from one or two labels, which are referred to as full *1-class* and *2-class non-IID*. The latter two are special cases of a uniform non-IID distribution ($q = 1$). Most of the backdoor defenses on FL, as well as approaches improving the aggregation function, focus on either 1-class and/or 2-class non-IID setups or the Dirichlet distribution with different main labels within the clients' datasets.

B. Poisoning Attacks on FL

In the past, FL has been shown to be vulnerable to so-called poisoning attacks. Such attacks can be untargeted [75], [10] to decrease the accuracy of the model and reduce the convergence speed of the global model [24], [91], [46], or *targeted*, also called *backdoor attacks* [7], [18], [8], [54], [30], which try to add additional functionality to a model while maintaining the main task accuracy (MA). All backdoor attacks have in common that there exists an *input trigger*, which is embedded within the raw data activating the backdoor. For a model that predicts labels for samples from a domain $\mathcal{X}$, the purpose of the backdoor is to make the poisoned model G_* predicting the *target label* $\mathcal{T}_\mathcal{A}$ when feeding a sample from the trigger set $\mathcal{I} \subset \mathcal{X}$ to the model. Therefore, the adversary $\mathcal{A}$ wants to achieve a high backdoor accuracy (BA) for samples from $\mathcal{I}$, as formalized in Eq. 2.

$$BA = \frac{|\{x \in \mathcal{I} \ : \ f(x, G_*) = \mathcal{T}_\mathcal{A}\}|}{|\mathcal{I}|} \tag{2}$$

Without further inspection, such backdoors remain undetected within the resulting global model and pose a danger to the model user. In FL, a poisoning attack can occur if one or more clients $C_i \in \{C_1, \ldots C_n\}$ are malicious and submit a manipulated local model to the server for compromising the aggregated model. Dependent on the attack algorithm and the attacker's capabilities, the adversary can manipulate the input data, the complete learning process including hyper-parameters, and the final weights of the trained local models to inject and hide a backdoor (with high BA while maintaining high MA). Furthermore, it can adapt to the defense by imitating the behavior of benign clients (e.g., scaling of the model weights) to stay undetected but still effective.

Different triggers have been proposed to activate the backdoors: 1) *Pixel backdoors* that get activated by a certain pixel pattern, like a red rectangle [7], [31], [53], which can also be distributed in fractions over multiple poisoned local models [92], 2) a *Label-Swap* backdoor that mislabels all samples of one class to a target class, or 3) a *Semantic Backdoor* that is activated when certain characteristics, e.g., a car in front of a striped background, is present in the input [7]. Examples of all triggers are shown in Fig. 1. We elaborate on these triggers in App. A. To achieve his goal, the adversary $\mathcal{A}$ chooses one or more of the following concepts:

Data Poisoning: $\mathcal{A}$ manipulates the training dataset $\mathcal{D}_i$, so that the resulting $\mathcal{D}_i^\mathcal{A}$ includes samples containing the trigger. To trade-off the effectiveness against the detectability of the attack, a respective ratio of malicious to benign samples, the so-called *Poison Data Rate (PDR)*, must be chosen by $\mathcal{A}$.

Model Poisoning: The adversary manipulates the training algorithm itself by changing hyper-parameters. Additionally, $\mathcal{A}$ can optimize against several objectives [22] in the form of additional loss functions (weighted by an α parameter) and constrain the loss(es) to stay as close as possible to benign behavior, especially if the adversary is aware of the defense. The weights of the resulting trained local models can be adapted to benign models to circumvent straightforward defenses that analyze the weights of all contributions and remove extreme outliers. This process conducted by an adaptive attacker is called *constrain-and-scale* [7].

C. Trusted Execution Environments

Trusted Execution Environments (TEEs) are programmable secure areas located within a processor that allow the execution of applications within secure enclaves, isolated from the remaining system. By using, e.g., memory encryption, access to them from outside the enclave, even from privileged processes, is restricted and thus guarantees the confidentiality of the enclaves' data. Attestation allows to verify the authenticity of the TEE and the integrity of code executed within the TEE [19]. Examples of TEEs are Intel SGX [19], AMD SEV [39], ARM TrustZone [69], and Nvidia Confidential Computing [68].

III. Problem Setting

In the following, we describe the considered system (Sect. III-A) and characterize the threat model (Sect. III-B).

A. System Setting

In the setup, we consider mainly cross-silo settings[2] with N clients $C_1, \ldots C_N$ that have private datasets $D_1, \ldots, D_N$ but will not share any data to prevent privacy leakages [1]. Further, we consider an aggregation server $\mathcal{S}$ that receives the individual models and aggregates them using FedAVG [56]. Aligned with recent work on poisoning attacks [6], [7], [65],

[2]Real-world scenarios and projects are, e.g. the FeTS project [1].

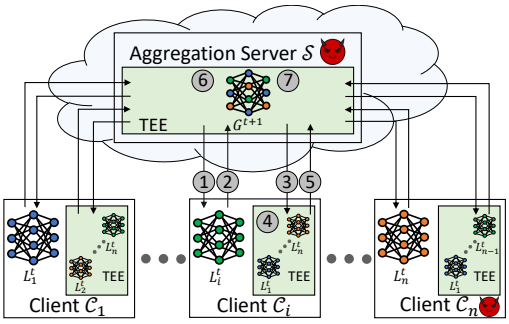


Fig. 2: Overview and steps of CrowdGuard.

we use an adapted version of the regular FedAVG algorithm and scale the local models' contributions equally with $1/n$ instead of weighting them based on their dataset sizes. This prevents malicious clients from artificially increasing their impact by reporting wrong dataset sizes. We keep the global learning rate constant at $\delta = 1$. In principle, multiple aggregation algorithms exist [10], [32], [61], [94], which either provide better performance or are more robust against byzantine contributions from local models. Our method can be used with different aggregation techniques since it is applied before the aggregation takes place.

We assume that each client and the server have an arbitrary TEE available, allowing the execution of code in the secure enclaves while isolating code and memory from the remaining system, including privileged parts. Thus, the TEEs shall prevent the remaining system from learning the data inside the enclave and therefore preserving the data's confidentiality. Further, the TEE needs to allow a remote machine to attest the code of the executed enclave. Depending on the application, e.g., in cross-silo applications where different institutions like hospitals collaboratively train a DNN, the machines performing the local training can be assumed to be powerful platforms, providing standard hardware features like TEEs and thus making this assumption reasonable.

An overview of the considered system is shown in Fig. 2, showing the clients, the aggregation server, and the individual TEEs (marked in green). Fig. 2 also shows the individual steps of our scheme, which we will discuss in Sect. III-D.

In contrast to existing work, we do not make any assumptions about the data distributions. Thus, the individual clients' data can follow the same distribution (IID), be distributed differently (non-IID), or even be disjoint.

B. Adversary Model

We consider two adversaries. The first, $\mathcal{A}$, aims to inject a backdoor into the FL system, while $\mathcal{A}^{\mathcal{P}}$ aims to learn information about the clients' data from the local model updates, violating the privacy of the data.

1) Poisoning Attacker $\mathcal{A}$: $\mathcal{A}^B$ (for the sake of brevity, denoted as $\mathcal{A}$) aims to manipulate the model that is resulting from the FL process and injects a backdoor into it by utilizing data and/or model poisoning (see Sect. II-B). If a certain, adversary-chosen trigger is present in the input (cf. Sect. II-B), the backdoor shall make the aggregated model G^{t+1} predicting an adversary-chosen target class $\mathcal{T}_{\mathcal{A}}$. From this goal, two objectives follow:

O1 - Attack Impact: To inject a backdoor successfully, $\mathcal{A}$ aims to make the aggregated model G^{t+1} predicting the backdoor target class $\mathcal{T}_{\mathcal{A}}$ for all trigger samples $\mathcal{I}$ from the input domain $\mathcal{X}$. Thus, its objective is to maximize the accuracy on the backdoor task, e.g., the BA.

If the server $\mathcal{S}$ notices the attack, it repeats the training process with a subset of clients until no backdoor is noticed anymore or filters the poisoned contributions. Therefore, a second objective for the adversary $\mathcal{A}$ is:

O2 - Stealthiness: Make the poisoned model updates inconspicuous such that $\mathcal{S}$ can neither identify the poisoned updates nor notices the performed backdoor attack[3].

From O2 also follows that the attack *must not* reduce the performance of the aggregated model on the main task (MA). If the predictions of the aggregated model G^{t+1} for a sample $x \in \mathcal{X}$ are denoted as $f(x, G^{t+1})$, G^{t+1} is the aggregated model without the poisoning attack, and G_*^{t+1} is the aggregated model including the poisoned contributions, then O1 and O2 result in the following goal of the adversary $\mathcal{A}$:

$$f(x, G_*^{t+1}) = \begin{cases} \mathcal{T}_{\mathcal{A}} \text{ if } x \in \mathcal{I} \\ f(x, G^{t+1}) \text{ if } x \notin \mathcal{I} \end{cases} \tag{3}$$

Aligned with previous work [10], [61], [65], [81], [45], we assume that $\mathcal{A}$ fully controls $n_{\mathcal{A}} < n/2$ clients in one round and overall $N_{\mathcal{A}} < N/2$ clients in the whole FL system.[4] Thus, it can freely manipulate their local datasets D_i, change the training process, or even manually change the submitted parameter updates and replace parameters with arbitrary numbers[5]. The ratio of poisoned models to all local models $n_{\mathcal{A}}/n$ is denoted as Poisoned Model Rate (PMR). Further, we assume that $\mathcal{A}$ knows all algorithms the server or clients execute. Thus, $\mathcal{A}$ can adapt its attack strategy and the client's behavior, e.g., hyperparameters and the objective of the local training, with respect to the deployed defense to make the attack inconspicuous (adaptive adversary).

2) Privacy Attacker $\mathcal{A}^{\mathcal{P}}$: The second adversary, $\mathcal{A}^{\mathcal{P}}$, aims to reconstruct information about the clients' local data. Aligned with existing work [6], [40], [65], we consider only privacy attacks that learn information about the clients' data by analyzing the local model updates. The aggregation of FL anonymizes the individual contributions, preventing $\mathcal{A}^{\mathcal{P}}$ from associating gained information with a specific client, and also smoothens the parameters. Thus, we will consider privacy attacks on the aggregated model out of the scope of this work.

In our threat model, we consider $\mathcal{A}^{\mathcal{P}}$ to be a malicious attacker that has arbitrary control over the aggregation server. Further, $\mathcal{A}^{\mathcal{P}}$ can control some of the clients to analyze any other client's local model that this client might receive. In contrast to existing work, the considered adversary $\mathcal{A}^{\mathcal{P}}$ even fully controls the server $\mathcal{S}$. However, the benign clients and the server can use remote attestation to verify the code and authenticity of the secure enclaves that are running on the server and the clients, respectively, before sharing local models.

[3] This differs backdoor from untargeted attacks, as the latter one can always be identified by a drop in the models' utility

[4] In each round n clients are selected for training out of all N clients.

[5] It should be noted that an adversary can manipulate the local dataset that is given to the client-side enclaves to manipulate their behavior.

3) TEE Security Assumptions: In the following, we consider arbitrary TEEs that isolate executed secure enclaves and allow a remote machine to attest the running enclave (cf. Sect. III-A). Thus, CrowdGuard is not restricted to TEEs of certain manufacturers. However, we assume that all used TEEs are trusted. Therefore, attacks on the used cryptographic algorithms and attacks that extract keys burned into the TEE are out of the scope of this paper.

Recently, several side-channel attacks have been proposed that extract data, e.g., the received models or cryptographic keys, from TEEs [13], [38], [87]. As discussed in App. A, with this model stealing or inference attacks could be executed. There exist already works to counter such attacks [9], [20], [12], [77]. Therefore, we consider attacks on the TEE architecture to be out of the scope of this work.

C. Requirements and Challenges

Based on the characterization of $\mathcal{A}$ and $\mathcal{A}^{\mathcal{P}}$, the following requirements for a backdoor defense can be derived:
R1: Prevent the backdoor attack, i.e., $\forall x \in \mathcal{I}: \ f(x, G_*^{t+1}) = f(x, G^{t+1})$.
R2: To be practical, the defense scheme must not reduce the benign performance of the resulting FL model, especially in the absence of any attack. Therefore, if no attack was performed and G^{t+1} is the aggregated model obtained using CrowdGuard, while $\hat{G}^{t+1}$ was obtained using plain FedAVG, then both model's outputs should be equal: $\forall x \in \mathcal{X}: f(x, \hat{G}^{t+1}) = f(x, G^{t+1})$.
R3: The defense must preserve the clients' privacy. Thus, the server $\mathcal{S}$ must not be able to access the models for running inference attacks. Nor should any other party, e.g., the clients, be able to run inference attacks on the models of other clients.

From these requirements, a number of challenges follow that CrowdGuard will address in the rest of the paper:

C1: How to effectively distinguish benign and poisoned models, especially for non-IID scenarios, to fulfill R1? Sect. IV explains how CrowdGuard can distinguish poisoned models and benign models, being trained on abnormal data.
C2: The server $\mathcal{S}$ must not be able to access the individual local models as this would enable $\mathcal{S}$ to run inference attacks (cf. R3). However, to identify poisoned model updates, $\mathcal{S}$ has to inspect the model updates L_i^t. A challenge that CrowdGuard will address is, therefore, how to inspect the local models without enabling any party to extract knowledge from them.
C3: CrowdGuard uses the predictions, including the hidden state outputs, of the local models on the local data of other clients for identifying a backdoor. However, the backdoor attack should not change the predictions for non-triggered input samples (cf. O2). Since it is unlikely that benign clients have many triggered input samples, a challenge that CrowdGuard will solve is how to use clients' local data for identifying the backdoor without having triggered samples.

D. Design of CrowdGuard

Due to the absence of validation data, existing approaches for backdoor mitigation on the aggregation server are restricted to using vector metrics or outlier detection [61], [81], [65]. These methods have limited effectiveness in non-IID settings or against sophisticated adversaries (cf. Sect. VII). To overcome these limitations, CrowdGuard involves sending the local models to the clients and collecting their feedback on the individual models to identify backdoored models. The use of such a client feedback-loop and a validation algorithm that analyzes changes in the outputs of the models' individual layers enables CrowdGuard to effectively identify poisoned models even in non-IID settings. Fig. 2 provides an overview of the individual steps of our approach, which are depicted in more detail in App. A. To protect the local models' confidentiality, all transmissions are encrypted, the code runs in TEEs, and each enclave is attested before receiving any model.

Each client begins with the setup of its secure-enclave, before receiving the global model from the server, trains its local model[6], and sends the model encrypted to an enclave running on the aggregation server $\mathcal{S}$ (steps 1 and 2 in Fig. 2). After collecting the individual model updates, $\mathcal{S}$ sends the local models to secure enclaves running on the clients $C_i \in C_1, \ldots, C_n$, which are utilized as validation clients (step 3 in Fig. 2). The validation is performed in a secure enclave to ensure confidentiality and prevent the client feedback-loop from increasing the surface for privacy attacks. Then, the client-side enclaves validate the models by analyzing changes in hidden layer outputs using the local datasets $\mathcal{D}_i \in \mathcal{D}_1, \ldots, \mathcal{D}_n$ (addressing C1), denoted as Step 4 in Fig. 2. Our method employs a novel metric called Hidden Layer Backdoor Inspection Metric (HLBIM) to analyze the obtained values iteratively and identify poisoned models based on statistical significance tests. The clients provide feedback to the server by voting for each model, indicating whether the local model is benign or suspicious (step 5 in Fig. 2). Afterward, the server applies a stacked clustering schema to combine the votes provided by the clients (step 6 in Fig. 2). This step mitigates manipulated votes from malicious clients who may provide manipulated data to the client-side enclave. Finally, the server removes the models marked as poisoned and uses a configured aggregation rule to aggregate the remaining models before sending the aggregated model G^{t+1} back to the clients for further training rounds (steps 6 and 7 in Fig. 2). It should be noted that although we focus on FedAVG as the aggregation rule in this paper, other rules such as Krum [10], trimmed mean [94], or median [94] can be used.

IV. CrowdGuard

In the following, we describe the details of CrowdGuard, starting with the overall architecture that allows utilizing clients' feedback in Sect. IV-A. Afterward, in Sect. IV-B, we describe the algorithm that is executed on the client-side of CrowdGuard to create the feedback and in Sect. IV-C, how the feedback of different clients is aggregated to be robust against manipulated feedback of malicious clients.

A. Privacy-Enhancing Architecture for Clients' Feedback

Given the impracticality of assuming the aggregation server to posses validation data [73], existing defenses are restricted to

[6]Depending on the respective application scenario, there are reasons for performing also the training inside a TEE, such as confidentiality of the training data, while there are also reasons against it, e.g., the computational overhead. CrowdGuard focuses on a backdoor resilient aggregation and supports both operational modes.

applying vector metrics, conducting outlier detection, or making predictions on randomly generated data that do not produce meaningful predictions. In addition, clients are reluctant to share their data with the server, since this would undermine the privacy advantage of FL, which allows for model training without data sharing. Secondly, transmitting large datasets to the server incurs significant communication overhead. However, sending models to the clients is also undesirable due to the risk of inference attacks by malicious clients ($\mathcal{A}^{\mathcal{P}}$) [36], [52], [63], [71], [76], [82], [95]. To overcome this dilemma, we propose a secure feedback loop that enables client-side validation of local models within TEEs (addressing C2). Additionally, server-side operations are also executed within a secure enclave to prevent abuse of the clients' feedback, e.g., to obtain information about their data. This guarantees the confidentiality of the processed data, including the models and corresponding layer outputs, even if $\mathcal{A}^{\mathcal{P}}$ has kernel privileges (fulfilling R3). Attesting the enclaves prior to any data transmission guarantees that the executed code does not leak the received models (cf. C2).

Setup Phase. During the setup phase, each client starts the secure enclave responsible for client-side validation and feedback provision to the server. Once the enclave is launched, the client shares its private dataset with the local enclave. Next, the server attests the integrity and authenticity of client-side enclaves. To mitigate the risk of fake validation requests from the server, which could potentially exploit clients' feedback to infer their data, clients also perform an attestation process to verify the correct execution of the FL server code. Notably, the relocation of the server to a secure enclave establishes a secure aggregation scheme. Hence, clients also attest the server-enclave during the setup phase.

Validation Phase. After completing the system setup, the FL process is initiated and the clients train their local models orchestrated by the server (represented by steps 1 and 2 in Fig. 2). After receiving the locally trained model from each client, the server distributes these models to the client-side validation enclaves (step 3 in Fig. 2). Each enclave utilizes its local datasets to identify any models that have been poisoned (step 4 in Fig. 2, for details on this algorithm, see Sect. IV-B) and submits this feedback to the server (step 5 in Fig. 2). It is important to note, that although the code of the validation enclaves is attested, the malicious clients can still manipulate their own validation result by providing manipulated validation data to the enclave. Hence, using secure enclaves on the client side only guarantees the models' confidentiality but not the votes' integrity.

To address this issue, a robust aggregation algorithm needs to be deployed on the server side to aggregate the feedback and remove manipulated feedback as well as noisy feedback from benign clients (see Sect. IV-C, step 6 in Fig. 2). Finally, the server uses the aggregated feedback to remove poisoned models, aggregate the remaining models and proceed with the next FL round (step 7 in Fig. 2).

B. Hidden-Layer Analysis for Backdoor Detection

To identify poisoned models using benign validation data, illustrated as step 4 in Fig. 2, CrowdGuard analyses the outputs of the individual layers of the DNN to distinguish between benign and backdoored (local) models. The result is provided as votes to the server afterward. This happens in

Algorithm 1 HLBIM Matrix Generation for client C_j

1: **Input:**
2: G^t, ▷ Global model of round t
3: L_i^t, ▷ All local contributions of round t including L_j^t
4: D_j ▷ Local dataset of client C_j
5: **Output:**
6: $\text{HLBIM}^{C/E}_{m_j\ m\ l}$ ▷ HLBIM matrices for Cosine & Euclidean distances
7: ▷ Generate deep layer outputs
8: $\text{DLO\_local}_{s\ m\ l} \leftarrow \{\}$
9: $\text{DLO\_global}_{s\ l} \leftarrow \{\}$
10: **for** s in D_j **do**
11: **for** m in L_i^t **do**
12: $\text{DLO\_local}_{s\ m\ l} \leftarrow$ deep_layer_outputs(s, m)
13: **end for**
14: $\text{DLO\_global}_{s\ l} \leftarrow$ deep_layer_outputs(s, G^t)
15: **end for**
16: ▷ Distance Generation
17: **for** $\text{dist}^{C/E}$ in [COSINE-distance; EUCLIDEAN-distance] **do**
18: $\text{DLO\_dist}^{C/E}_{s\ m\ l} \leftarrow \text{dist}^{C/E}(\text{DLO\_local}_{s\ m\ l}, \text{DLO\_global}_{s\ l})$
19: ▷ Scale relative distances to HLBIM
20: **for** $\text{dlo}^{C/E}_{s\ m\ l}$ in $\text{DLO\_dist}^{C/E}_{s\ m\ l}$ **do**
21: $\text{dlo\_rel}^{C/E} \leftarrow \text{dlo}^{C/E}_{s\ m\ l}$ / $\text{DLO\_dist}^{C/E}_{s\ m_j\ l}$
22: $\text{DLO\_squared}^{C/E}_{s\ m_j\ m\ l} \leftarrow |\text{dlo\_rel}^{C/E}\text{-}1| * (\text{dlo\_rel}^{C/E}\text{-}1)$
23: **end for**
24: $\text{DLO\_avg}^{C/E}_{lab\ m_j\ m\ l} \leftarrow$ AVG(labels lab, $\text{DLO\_squared}^{C/E}_{s\ m_j\ m\ l}$)
25: $\text{HLBIM}^{C/E}_{m_j\ m\ l} \leftarrow$ CONCAT(labels lab, $\text{DLO\_avg}^{C/E}_{lab\ m_j\ m\ l}$)
26: **end for**

two steps: 1) extraction of a novel metric, the Hidden Layer Backdoor Inspection Metric (HLBIM), from the local models to inspect them for backdoors and 2) analyzing the HLBIM via probabilistic tests to produce voting decisions.

HLBIM Motivation Analyzing two metrics with statistical tests enables CrowdGuard to detect different model manipulation strategies. Vectors (i.e., DNN's parameters) can be manipulated in two ways: Without changing direction, which is the orientation of the vector (checked by Euclidean distance) or with changing direction (increased Cosine distance) to the previous vector state. Using both metrics enables the detection of malicious model changes (addressing C3). HLBIM carves out significant changes in the plain values of both distances. The calculation of the HLBIM is explained in the following.

HLBIM Matrix Generation As depicted in Alg. 1 lines 1-6, the global model G^t, the local models $L_i^t \in \{L_1^t, \ldots, L_n^t\}$, and the validation client's local data D_j are used to generate two HLBIM matrices based on Cosine (HLBIM^C) and Euclidean (HLBIM^E) distances. Both matrices are based on the deep layer outputs (DLOs), which can be obtained by feeding the local data into both, the local models and the global model. During inference, we keep track of the outputs of each sample, each model, and each layer, depicting one DLO in a respective matrix (lines 7-15). Notably, CrowdGuard analyzes all layers' DLOs. Otherwise, if CrowdGuard only considered a subset of layers, the adversary could utilize the unconsidered layers for injecting the backdoor without being detected. To prevent such an attack, the DLO matrices are based on all layers.

The two DLO matrices for Euclidean and Cosine distances are then used to calculate the final HLBIM within five steps (lines 18-25): 1) Distances between the global models and each local model are computed for each DLO (line 18). In these distances, the backdoor behavior is not yet detectable with high significance. 2) A ratio of the DLOs is generated, using the validating client's local model as a reference to highlight differences between the models regarding a reference model (line 21). The rationale is that each client assumes his

Algorithm 2 Voting Decision via Model Pruning for validation client V_j

```
1: Input:
2: HLBIM^{C/E}_{m_j m l}                          ▷ HLBIM matrices for Cosine & Euclidean distances
3: Output:
4: client_voting                                  ▷ Binary vector with client decisions for each L_i^t
5:                                                ▷ Analyze HLBIM via dimension reduction
6: client_voting_model_is_benign ← Array of |i| ones
7: for dist_type^{C/E} in [COSINE-distance; EUCLIDEAN-distance] do
8:     significant ← True
9:     pruned_models ← {}
10:    while significant do
11:                                               ▷ Filter already pruned models
12:        HLBIM _pruned ← HLBIM^{C/E}_{m_j m l} ∀ m ∉ pruned_models
13:                                               ▷ Analyze remaining models
14:        pc_dim1_values ← PCA(HLBIM _pruned)[0]
15:        significant ← SIGNIFICANCE(pc_dim1_values)
16:        malicious_models ← {}
17:        if significant then
18:            clusters ← AGGLOM(nclusters = 2, pc_dim1_values)
19:            malicious_models ← MIN_CLUSTER(clusters).models()
20:        end if
21:        pruned_models.add(malicious_models)
22:                                               ▷ Safety Abort Criterion
23:        if |pruned_models| > FLOOR(|L_i^t| -1) / 2 then
24:            malicious_models.remove(MIN(malicious_models))
25:            significant ← False
26:        end if
27:        client_voting_model_is_benign[malicious_models.indices] = 0
28:    end while
29: end for
```

own model to be benign, while the ratio highlights already small differences between the model updates. If the DLOs are equal, the ratio will be one. 3) To further highlight the differences between the ratios, the values are scaled by subtracting one and squaring the result while retaining the sign (line 22). 4) The resulting DLO matrices are averaged over the sample dimension for each label, thereby carving out the effect for each label class separately, with the first matrix index changing from sample s to label lab (line 24). 5) To reduce the matrix dimension without losing information and thus saving computational costs in subsequent stages, the lab dimension is flattened by concatenating the values (line 25). It is important to keep the values from different labels separate until the pruning. This separation ensures that only samples from the same label, which traverse the model similarly, are used by our method to identify abnormal behavior. Averaging beforehand would lead to the loss of important information and limit the effectiveness of the metric.

Voting Decision via Model Pruning To detect poisoned models, the HLBIM matrices must be analyzed.[7] As described in Alg. 2, CrowdGuard first leverages a Principal Component Analysis (PCA) on the HLBIM matrix[8] to highlight the differences between different models (cf. line 14), before iteratively pruning the poisoned models.

The PCA reduces the two-dimensional matrix (models $m \times$ layers l) to a single dimension by analyzing the Principal Component (PC) values of the first dimension (pc_dim1_values in Alg. 2). To detect the presence of backdoored models, we rely on statistical significance tests on the first PCA dimension.[9] The intuition is that the median PC value is benign (cf. Sect. III-B). If all models are benign, the PC values follow a similar distribution and with this, the absolute distances to the median value of the PC values *above* and *below* the median value should follow the same distributions. A visualization of such two distributions is shown in Fig. 3, where the blue and dark red lines represent the two mentioned distributions.

To compare and identify significant differences between these distributions, we analyze their *means*, *variances*, and *outliers* leveraging different significance tests (line 15 in Alg. 2). Outliers are considered since in general the mean and the variance of a distribution is not necessarily affected by few outlier samples. First, CrowdGuard forces an equal mean via Student-T-Test [55]. However, the variance can differ significantly even for equal means, allowing an adversary to adapt the backdoors according to these metrics. Therefore, CrowdGuard checks for matching variance via F-Test[10].

To enhance robustness and prevent the adversary from attempting to fool the F-Test, we also employ the D-Test (Kolmogorov-Smirnov) to analyze the overall distributions. This additional test ensures equal goodness of fit of the distributions. We set a significance level of 0.01 for each test. [11] Upon passing these tests, we investigate outliers that might not influenced the former metrics. These outliers represent weakly hidden poisoned models that would have a high impact on the aggregated model. To identify such outliers, we set two thresholds: 1) We analyze the interquartile range of all data points by using a boxplot. 2) We analyze the distance of each point regarding the interval spanned by the 3σ-rule [70]. Data points lying outside the interval are marked as significant.[12] The algorithm is provided in App. A.

If the tests indicate the presence of poisoned models, we employ hierarchical agglomerative clustering [67] to generate two clusters and prune the models located in the smaller one (lines 16-21)[13]. This pruning is repeated until the significance tests report the absence of suspicious models (line 15)[14]. Due to the iterative pruning approach, we can detect and remove different backdoors, within one FL round t, that would not be detectable all together in a single cluster. The iterative pruning approach enables us to detect and remove different backdoors within a single federated learning round, which may not be detectable as a whole in a single cluster.

One pruning sequence is visualized in Fig. 3b-Fig. 3d. In Fig. 3d, no more significant models are detected. Another example can be seen in Fig. 4e to Fig. 4g.

[7] Euclidean and Cosine HLBIM matrices are analyzed independently.

[8] The first dimension m_j of $\text{HLBIM}^{C/E}_{m_j\ m\ l}$ is fixed to index j of the validating client V_j. Therefore the matrix is two dimensional.

[9] App. A discusses, that using multiple PC dimensions is suboptimal.

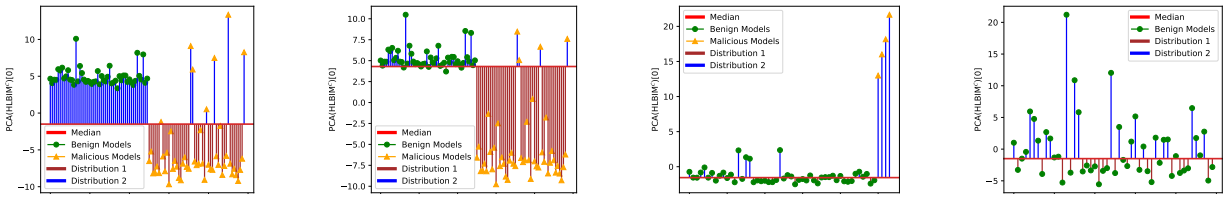
[10] Levene-Test [50] to asses equal variances.

[11] Typically, statistical tests use a significance level of 0.05. By using a lower p-value, we aim to reduce the likelihood of False-Positives and increase the sensitivity of CrowdGuard.

[12] The boxplot focuses on detecting single outliers not violating the 3σ-rule [70], while the 3σ-rule outperforms the boxplot for multiple outliers whose distance to the mean grow increasingly.

[13] Notably, the median is not suitable to separate both groups as it would always split the data points into two equally-sized groups. However, if one of them is discarded but significantly less than 50% of models are poisoned, this will result in many FPs.

[14] Additionally, as a fine-tuning step, we add an abort criterion to stop the pruning process if more than $\frac{n}{2}$ -1 models have been pruned. This criterion is included to prevent obvious False Positives in a small number of experiments. While this may theoretically allow for False Negatives, i.e., undetected poisoned models due to the abort criterion coinciding with False Positives, our experiments (see Sect. V) show that the backdoor leads to significant differences within certain DLOs even for benign input samples. Consequently, the PC values of poisoned models differ more from the median (benign) value compared to benign models, ensuring that malicious models are consistently identified first.



(a) Pruning round 1 (b) Pruning round 2 (c) Pruning round 3 (d) Pruning round 4

Fig. 3: Visualized distributions generated by Alg. 4 for pruning CrowdGuard with $n = 100$, $PMR = 40\%$. Distributions are considered to differ significantly, indicating backdoors found from (a) - (c). Poisoned models are pruned iteratively.

C. Voting Aggregation

Stacked Clustering: After the clients provided their votes in the form of binary vectors to the server[15], the server is confronted with potentially malicious votes from adversaries as well as with unintentional wrong votes of benign clients, which can occur if a model exceeds, i.e., an outlier threshold slightly. To address this issue, we employ a two-level stacked clustering, that selects the most representative voting vector from all submissions. The purpose of the first level is to eliminate obvious malicious votes by pruning the smaller of two clusters, as due to the majority assumption the larger cluster has to be the benign one. The second clustering is a plain-majority-voting on the rest, which are expected to be mostly benign votes. Thus, this step ensures the robustness of the aggregation against minor misclassifications of benign clients and adversaries, deviating only slightly from benign votes to remain inconspicuous.

In comparison, for plain majority voting, in a scenario with 49% adversarial clients who vote for all malicious models to be benign, a single incorrect vote of one benign client for a poisoned model can result in the acceptance of this model.

As depicted in Alg. 3 in detail, the server first generates two clusters on the binary voting vectors by agglomerative clustering [67] and identifies the bigger one as votes from benign clients, which is reasonable due to the majority assumption (cf. Sect. III-B). However, this cluster can contain minor errors of benign clients as well as malicious clients that manipulate their voting to be similar to benign behavior. Therefore, we conduct a second clustering to extract the most frequent binary voting vector by using DBSCAN [23] and inspecting the cluster sizes of the output. The voting of the biggest cluster is the final result of the voting aggregation.

[15]It is worth noting, that each client does not evaluate its own local model, but just reports it as benign by default.

Algorithm 3 Voting Filtering via Stacked Clustering

```
Input:
voting_matrix                          ▷ Matrix of client votes. Dimensions: (|C_i| × |L_i^t|)
Output:
aggregated_voting                      ▷ List with final voting decisions for each L_i^t
                                       ▷ Majority Cluster Detection
clusters ← AGGLOM(nclusters = 2, voting_matrix)
majority_cluster ← MAX_CLUSTER(clusters)
                                       ▷ Miss-Classification Compensation
filtered_cluster ← DBSCAN(majority_cluster, min_samples=1, ϵ=0.5)
aggregated_voting ← {"all-benign"}
max_count ← 0
for cluster in filtered_cluster do
    count ← |cluster|
    if count > max_count then
        aggregated_voting ← cluster[0].decision
        max_count ← count
    end if
end for
```

Robustness: The stacked clustering ensures robustness in scenarios where every malicious client marks every benign model as malicious and vice versa, since these manipulated votes are removed after the first clustering. Malicious feedback, where malicious clients vote as benign as possible, trying to invert the decision for one specific model, will be mitigated by the second clustering. The same holds for minor voting errors of benign clients. Leveraging the stacked clustering approach, which relies on the majority assumption, the algorithm remains robust against PMRs of up to 49%.

This strategy of first identifying benign votes by majority and then selecting the best voting via majority as the final decision outperforms naïve majority voting, which would not ignore False-Positives of benign clients, making it less robust. We discuss respective experiments in App. A.

V. Evaluation

In this section, we first depict our experimental setup in Sect. V-A and then describe the influence of various parameters in Sect. V-B. Afterward, in Sect. V-C, we investigate the runtime performance of our approach.

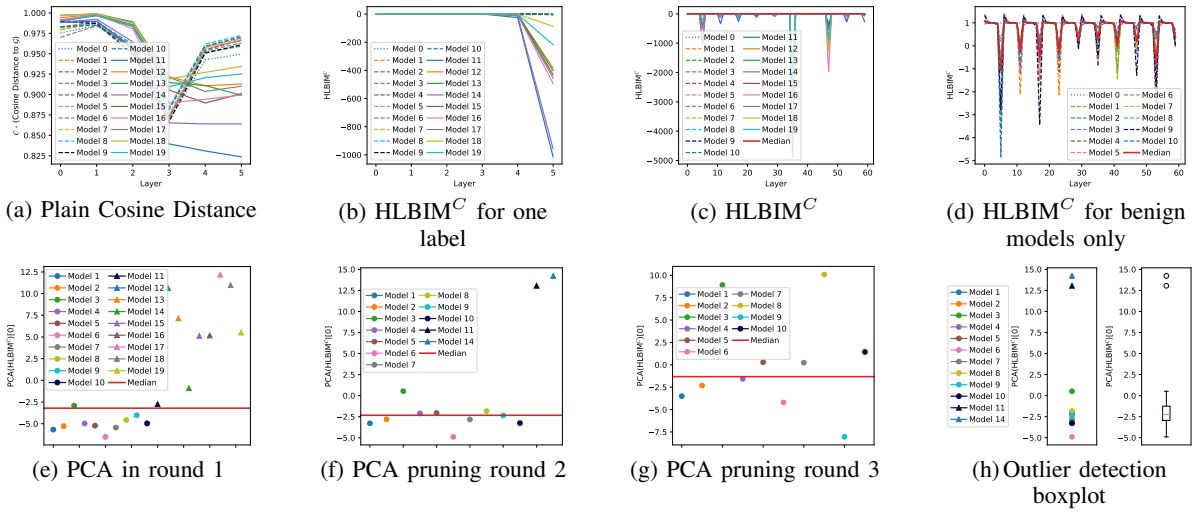
A. Experimental Setup

To simplify the comparison of our evaluation with other poisoning defenses, we aligned our experimental setup with recent works [7], [14], [73], as we describe in the following.

Computational Setup: All experiments were implemented in Python using the Deep Learning library PyTorch [5]. The experiments were executed on a server with an Intel Xeon 5318S with Intel SGXv2, 2 Nvidia RTX A6000, and 512 GB main memory, from which 128GB were reserved as secure memory. For executing Python code inside an SGX enclave, we leveraged the library Gramine [83].

Datasets: For our evaluation, we use the popular benchmark datasets CIFAR-10 [43], consisting of 50k training images and 10k test images, and the MNIST [21] dataset, consisting of 60k training images and 10k test images. Both datasets contain samples from 10 classes and are frequently used for evaluating poisoning defenses [7], [15], [28], [56], [61], [65], [73]. To simulate the FL setup, we split the training dataset into local datasets $D_k \in \{D_1, \ldots, D_N\}$ consisting of 2 560 samples[16], one D_k for each $C_k \in \{C_1, \ldots, C_N\}$. Aligned with

[16]The dataset size of 2560 samples is chosen based on other approaches (between 600 [7], [96] and 2000 [81]) to ease the comparison.



(a) Plain Cosine Distance (b) HLBIM^C for one label (c) HLBIM^C (d) HLBIM^C for benign models only

(e) PCA in round 1 (f) PCA pruning round 2 (g) PCA pruning round 3 (h) Outlier detection boxplot

Fig. 4: Visualization of intermediate outputs of CrowdGuard with default configurations (models 0 to 10 are benign).

existing work [14], [73], we created datasets for different non-IID scenarios: 1) For 1-class non-IID with a non-IID rate q, a main label is chosen randomly for each client and q percent of samples in D_i are changed to samples with the respective main label, while the remaining labels are chosen from all labels uniformly distributed. Therefore, for a non-IID rate of $q = 1.0$, each client only uses data from its main label(s), such that the data of different clients are disjoint if they have different main labels. 2) For 2-class non-IID we do the same but choose the subsequent label of the main label as "second main label". 3) For Dirichlet and Normal distribution, we produced label counts according to the respective distribution.

For the CIFAR-10 dataset, we used a light version of the Resnet-18 network, as described by Bagdasaryan *et al.* [7], which delivers five deep layers and one final layer output. For MNIST, we reimplemented a version of the Convolutional Neural Network (CNN), following Cao *et al.* [15].

Default Configurations: The default parameters and configurations, including hyperparameters for our experiments, are provided in Tab. IV in App. A. In our default setup, we use the CIFAR-10 [43] dataset, an adaptive adversary leveraging a *constrain-and-scale* attack [7] to inject a semantic backdoor, a Poisoned Data Rate (PDR) of 0.1, a PMR or 0.45 and set α to 0.7. We utilize $|C_i| = 20$ clients to participate in the FL round t as well as in the feedback loop. We select the main label of each client according to its index i, so that we have the most disjoint label settings and prevent getting multiple clients with the same main label by chance.

B. Outputs and Influence Factors

In Sect. V-B1, we visualize and explain the output of our experiments and list all configurations. Afterward, we discuss the parameters and other influencing factors in Sect. V-B2.

1) Experiment Outputs: To improve the comprehension of our approach, in Fig. 4 we visualize the intermediate outputs of our algorithm. The experiment is conducted with our default configurations from Tab. IV and therefore contains 20 clients. Fig. 4a depicts the plain Cosine distance from each local model L_i^t to the global model G^t for one label (in this case, label 8) averaged over that label. As explained in Sect. IV-B, from this metric alone, one cannot clearly identify the poisoned models. To enable backdoor detection, we produce the matrices containing our novel metric HLBIM, which can be seen for Cosine distance in Fig. 4b for one label. Subsequently, in Fig. 4c, we can then observe the whole HLBIM^C plot. The malicious models are responsible for the peaks. This can be analyzed by comparing the values for HLBIM^C to the ones in the poisoned-free version in Fig. 4d. According to Alg. 2, we conduct the PCA on the HLBIM matrices to obtain Fig. 4e, which is used in the first pruning round of our significance test. The results of the second and third rounds of pruning are depicted in Fig. 4f and Fig. 4g. At this step, we end up with just benign models, which results in negative significance tests. Fig. 4d shows the cleaned HLBIM^C graph used to produce the last PC values in Fig. 4g. Fig. 4h shows an exemplary boxplot of our outlier detection algorithm during the second pruning round (cf. Fig. 4f). The abnormalities in the malicious models derive from their local BA, which in our experiments is always 100%. This means the attacker is able to incorporate the backdoor in his local model.

Conducted Experiments: Tab. I lists all of our conducted experiments, each changing one parameter of the default configuration. The used metrics are defined in App. A. Additionally, we tested the defense against two untargeted poisoning attacks: 1) We randomly selected the labels for each sample in the training and test set. 2) We changed the learning algorithm to maximize the loss function. Both experiments delivered 100% defense success rate. Thus we can claim that CrowdGuard also reduces the risk of *untargeted poisoning attacks*. Furthermore, the following two combined attacks that integrate two backdoors with different triggers and target labels $T_{\mathcal{A}}$ at once have been evaluated and results are reported in Tab. I. The following two scenarios were considered: 1) Every $\mathcal{A}$ attempts to inject both backdoors. 2) Half of the malicious clients integrate one backdoor, and the other half integrate a different backdoor.

2) Influence Factors: In App. A we evaluate different factors that might affect the performance of CrowdGuard (α, the PDR, and the distribution of local data). However, in all cases CrowdGuard effectively identifies benign and poisoned models independently from the scenario. Thus, it achieves 100% True-Positive-Rate (TPR) as well as True-Negative-Rate (TNR). Further, we observe that CrowdGuard has no negative impact on the MA (fulfilling R2). The same perfect detection rates are achieved in our other experiments, listed in Tab. I.

Disjoint Data Scenario: We conducted an experiment with only six benign and four malicious clients in 1-class non-IID for $q = [0.0]$ and assigned samples from each of the ten existing labels in the dataset to a different client. This setup results in completely disjoint training data, which reflects a full non-IID scenario. CrowdGuard was also effective in the detection of the four malicious clients even in this edge-case scenario, fulfilling C1.

Besides that, we tested a scenario, where all 20 clients were benign and changed this setting to a 1-class non-IID scenario, where 40% of clients possess the same main label, e.g., we assigned only red cars from CIFAR-10. We observed that the defense did not falsely filter out benign models and hence such a scenario does not negatively affect the convergence.

MNIST: In addition, we evaluated CrowdGuard on the MNIST dataset in 1-class and 2-class non-IID with $q = [0.0, 1.0]$ scenarios starting from FL round 100 using the Label Swap backdoor and also for a smaller learning rate of 0.001 at the malicious clients. This showed that our approach is not restricted to one specific dataset, thus not hindering deployment in real-world scenarios.

Varying PMR: Our experiments are conducted with the biggest Poisoned Model Rate (PMR) possible regarding the C_i clients and, since we are pruning the malicious models, a smaller PMR would still result in the same outcome. Hence, we can conclude that the PMR has no influence. In the default setting, the PMR has a maximum value of $^9/_{11} = 45\%$. To stress this parameter to the maximum, we conducted an additional experiment with $n = 100$ clients and a PMR of 49%. The results also show perfect detection rates of 100%, making the approach effective in small and large FL setups.

Randomly Initialised Model: The only exception with regard to CrowdGuard's effectiveness occurred in the first round ($t = 0$) when starting with a randomly initialized model. Here, CrowdGuard accepted all models, including the poisoned models. However, as the model's parameters of all models were changed significantly, the impact of the poisoned models was negligible. Therefore, the BA remained 0% and the attack was not effective, while the adversary was not able to inject the backdoor in later rounds ($t \geq 1$). This experiment showed two facts: 1) It is harder for an attacker to implement the backdoor as long as the model did not converge to a certain MA, as already discussed in earlier work [7]. 2) Nevertheless, CrowdGuard already detected 100% of malicious models in round $t = 1$, meaning our approach is not limited to already converged global models G^t.

Robustness against Adaptive Adversaries: An adversary can adapt to the defense in various ways by integrating a second loss function (cf. Bagdasaryan *et al.* [7]). Thereby, the adversary can try to minimize the distance of the model weights to the global model G^t or first train a benign local model and then try to adapt precisely to our defense algorithm by leveraging DLOs measured on that model. We conducted experiments with both adaptation strategies, finding that the former delivered better results for the adversary, meaning that poisoned models were harder to identify by CrowdGuard (regarding the significance level). Thus the more difficult-to-handle adaptation method is part of our default setting but does not prevent CrowdGuard from detecting the backdoor. The most relevant reason for that is, that the adversary cannot adapt to the other clients' local data. If the adversary increases the level of adaptiveness extremely, he fails in introducing the required BA in the local model, so that his contribution is averaged out, even without clipping or noising methods (cf. adversarial's dilemma in Sect. II-B), fulfilling R1.

As the client-side validation algorithm compares the PC values of the individual models against each other, a sophisticated adversary might try to adapt its attack by splitting all malicious clients into different groups leveraging different PDRs. To show CrowdGuard's robustness in such scenarios, we conducted an experiment where two groups of malicious clients use different PDRs of 10% and 30%. However, the iterative running enabled CrowdGuard to identify all benign and poisoned models effectively.

Overall, CrowdGuard is robust in various scenarios, independent of specific FL system settings, and therefore applicable in real-world scenarios. The reasons for the high success rates are that the usage of benign validation data allows a detailed analysis of the local models' behavior. In addition, benign clients rarely deliver wrong votes due to our significance test (cf. Alg. 4) based on HLBIM. Even if the malicious clients manipulate the voting of its secure enclave, they are compensated by the subsequent stacked clustering voting aggregation.

C. Runtime Overhead of CrowdGuard

To analyze the performance overhead introduced by CrowdGuard, we measured the runtime of the different phases of CrowdGuard's client-validation in a TEE (SGX) and compared it to the execution outside of a TEE on the CPU for our default setting with 20 models. Further, we also measured the runtime when using the GPU for the predictions. The results, being averaged over ten executions, are shown in Tab. II. As the table shows, the overhead for the attestation is with 0.1s in average negligible, as well as the time for the pruning. The HLBIM calculation takes similar time for all three versions, as it is done for all three platforms on the CPU, also if an accelerator is available. The main difference between the individual versions is the time that is needed for predicting the DLOs. Here, using an accelerator, i.e., a GPU, shows a significant performance improvement. We elaborate on the runtime overhead in Sect. VI-C.

D. Comparison with Existing Approaches

Tab. III shows the comparison of CrowdGuard with several state-of-the-art IR approaches [28], [62], [96], as well as DF methods [10], [81], [94] in our default scenario. Notably, existing approaches make certain assumptions about the attack strategy and data scenario: Zhao *et al.* assume that the attack

Analyzed Parameter	Parameter Values	TPRs	TNRs	FPRs	FNRs
Data distributions	CIFAR-10, 1-class non-IID, $q = [0.0, 0.1, ..., 1.0]$	100%	100%	0%	0%
	CIFAR-10, 2-class non-IID, $q = [0.0, 0.1, ..., 1.0]$	100%	100%	0%	0%
	CIFAR-10, Dirichlet	100%	100%	0%	0%
	CIFAR-10, Normal	100%	100%	0%	0%
	MNIST 1-class non-IID, $q = [0.0, 1.0]$	100%	100%	0%	0%
Adversarial adaptation rate α	$\alpha = [0.1, 0.2, ..., 0.9]$	100%	100%	0%	0%
Poison Data Rate (PDR)	$pdr = [0.1, 0.2, ..., 0.9]$	100%	100%	0%	0%
Poison Model Rate (PMR) & number of clients n	$pmr(n = 20) = [0.05, 0.1, ..., 0.45]$	100%	100%	0%	0%
	$pmr(n = 100) = [0.01, 0.2, ..., 0.49]$	100%	100%	0%	0%
	$pmr(n = 10) = [0.1, 0.2, ..., 0.4]$, (1-class non-IID $q = 0.0$)	100%	100%	0%	0%
Poisoning	Pixel Backdoors , Label Swap, Semantic	100%	100%	0%	0%
	2 combined attacks	100%	100%	0%	0%
	2 untargeted attacks	100%	100%	0%	0%
Starting FL round t	$t = 1000$	100%	100%	0%	0%
	$t = 0$	100%	0%	100%	0%
	$t >= 1$	100%	100%	0%	0%
Malicious Training Learning Rate(M-LR)	$LR = [0.01, 0.001]$	100%	100%	0%	0%

TABLE I: Listing of conducted experiments with TPR, TNR, FPR, and FNR of CrowdGuard. The default settings are used and only the analyzed parameter is changed for one experiment. Multiple parameters within brackets denote multiple experiments.

Platform	Attestation	Model Transmission	Predictions	HLBIM	Pruning	Total
SGX	0.1	4.0	19.7	5.5	0.1	29.5
CPU	-	2.6	10.5	5.0	0.0	18.1
GPU	-	2.7	4.8	5.1	0.0	12.8

TABLE II: Average evaluation times of CrowdGuard's steps in SGX, outside SGX on a CPU and on a GPU in seconds.

Approach	BA	MA	TPR	TNR	PRC
No Attack	0.0	62.0			
No Defense	80.0	61.5			
Differential Privacy [62]	80.0	50.6	-	-	-
Zhao *et al.* [96]	100.0	61.2	-	-	-
Median [94]	0.0	10.0	-	-	-
FoolsGold [28]	0.0	10.0	100.0	9.0	47.4
Krum [10]	100.0	63.8	88.9	0.0	42.1
Auror [81]	80.0	68.4	0.0	100.0	-
CrowdGuard	0.0	62.0	100.0	100.0	100.0

TABLE III: Experiment results of CrowdGuard and six state-of-the-art defenses (three IR-based and three DF-based approaches) for the CIFAR-10 datasets in terms of Backdoor Accuracy (BA), Main Task Accuracy (MA), True-Positive-Rate (TPR), True-Negative-Rate (TNR), and Precision (PRC) for one FL round t, all values in percentage.

reduces the MA [96], which does not always hold in practice (cf. O2) and hence, hinders detection. Median [94], instead, effectively mitigates the attack but cannot handle the non-IID scenario and drops the MA to the performance of a naïve classifier. Existing DF approaches make assumptions about the attack strategy and data distribution: Auror [81] clusters the parameters of all model updates into two clusters once and considers the smaller cluster as suspicious. This approach fails for the highly non-IID scenario that we described in Sect. II, showing the advantage of CrowdGuard's iterative pruning. Krum assumes the benign models to have low distances among each other [10], which does not hold for the non-IID scenario and can be easily circumvented by an adaptive adversary (cf. Sect. V-B2), while FoolsGold [28] cannot handle the case that some benign clients have similar data, thus having updates that point in the same direction. In comparison, CrowdGuard does not make any assumptions about the data scenario or the attack strategy while still being able to identify all poisoned and benign models correctly.

It should be noted that Tab. III showing our default scenario contains non-IID data distributions (disjoint data). As each client optimizes for its own training set, few benign models can negatively affect the MA on the test set containing all labels. In this special case the MA can improve by excluding benign models. For example, Krum selects one local model as new global model, which might have superior performance on the main task in the test set, but cannot prevent the backdoor from being active.

In a separate experiment, we also evaluated BaFFLe [6] on CIFAR-10 data. Notably, BaFFLe first needs a benign warm-up phase without attacks [6][17], which we set to 30 rounds starting from FL round 100. It is necessary to evaluate multiple rounds to analyze BaFFle's ability to accept a benign aggregation model. This ensures that BaFFle does not reject every update. Therefore, we performed this comparison in a separate experimental setting using a less converged model, hence these results are not included in Tab. III. We observed that BaFFLe was not resilient to the constrain-and-scale attack [7], resulting in a BA of 80%, while CrowdGuard effectively identified also in this setting benign and poisoned models (TPR=100%, TNR=100%, BA=0%). CrowdGuard is more efficient than BaFFLe, because CrowdGuard analyzes the outputs of all layers, while BaFFLe focuses on the output of the last layer, which an adversary will always try to make inconspicuous (O2).

[17]We discuss the real-world applicability of this precondition in Sect. VII.

E. Alternative Aggregation Techniques

As described in Sect. II-A, we focus on a version of FedAVG for aggregation that weights all accepted clients' models equally, regardless of their dataset sizes. To show the general applicability of CrowdGuard, we conducted an experiment where we weighted the accepted models' based on the dataset sizes which were reported by the clients. We observed the same performance as in our default setting (TPR=100%, TNR=100%) and the aggregated model achieved a MA of 62%. Notably, the ability of CrowdGuard to filter poisoned models is independent of the concrete aggregation function, as the aggregation is performed after the filtering process is finished.

VI. Discussion

In the earlier sections, we introduced CrowdGuard, a novel defense against backdoors in FL that is fully compatible with secure aggregation techniques. In the following, we will discuss its security, parameters, as well as its limitations.

A. Parameterization

In our significance-based algorithm (cf. Alg. 4), we introduce thresholds in the form of significance levels, that function as parameters for CrowdGuard. The p-value of the probabilistic tests is set to 0.01 and the outlier thresholds are dynamic values based on the observed data. Thus, we purely rely on probabilistic thresholds and do not include empirically determined limits, that are dataset-dependent. We demonstrated that those parameters paired with Alg. 3 are robust against (un)targeted poisoning attacks. Naturally, higher p-values might stop the iterative validation although anomalous values are still present, hence increasing the probability of FNs, and, conversely, lower p-values make it more sensitive toward outliers leading to more rejected benign models and an increased FPR.

B. Necessity of TEEs

CrowdGuard requires the availability of a TEE on the server side and at clients. In the considered cross-silo scenario, where multiple larger computation centers collaborate on training a DNN, the availability of TEEs is reasonable to assume. While not all devices currently have TEEs, many of today's mobile devices possess TEEs (e.g.,ARM-TrustZone). Although their deployment is restricted to vendors, this might change in the future. In scenarios with TEE-less clients it would be possible to, e.g., select only a subset of all clients for the feedback loop. Notably, in this case it is no longer guaranteed that the majority of the selected clients will be benign. In comparison, if all clients are used, this is guaranteed by the underlying threat model Sect. III-B. Our analysis provided in App. A shows the probability of selecting a malicious majority in such a scenario. We note that for smaller PMRs it is negligible even if only 50 out of 1000 clients are selected for validation. Therefore, in scenarios with TEE-less clients, stronger assumptions about the maximal PMR are necessary.

C. Computational Overhead

Overhead of Computations. Validating the individual local models introduces an additional performance overhead. In Sect. V-C, we evaluated this overhead and measured the runtimes of the individual phases of CrowdGuard. Although the total runtime of 25.5s in average seems to be acceptable given much longer time needed for training a model, this overhead can still be further reduced. The major part of the overhead is created by the prediction of the DLOs. One strategy to improve the performance of CrowdGuard would be utilizing ML accelerators, that include TEEs [68] or TEEs, that expand their security guarantees to accelerators [85], [98]. Also, the clients could decide to use only a representative subset of their local dataset for the validation. Other possible strategies to further optimize the runtime performance include parallelizing the calculations for the different models in each step, e.g., calculating the distances between the local models' and global model's DLO for different local models in parallel. Further improvements could be achieved by using a more performance-oriented language than Python. However, as the focus of this paper is on the design of a defense against poisoning attacks in FL, we consider those optimizations to be out of the scope. Therefore, it is left to future work to optimize the runtime performance of CrowdGuard.

Memory Overhead Regarding memory, CrowdGuard needs to hold the parameters of one DNN, the predicted DLOs of two DNNs, the DLOs for the global model, and for the one local model at the same time. In our setup for CIFAR-10 these aggregate to 94 218 float-numbers per sample. After calculating the local model's DLOs, the distance to the global model DLOs can be determined and only these distances for each layer, i.e., a single number for each layer, need to be stored when continuing to process the next model. Depending on the available system resources, models that were not processed so far can be either stored (encrypted) on the file system, or the server sends the models sequentially to the clients. Hence, memory might be a limitation of CrowdGuard in some TEEs. Nevertheless, newer architectures, such as SGXv2 [47], also provide large amounts of memory within an enclave.

Overhead of Client-Side Validation. The other aspect that causes the overhead is the distribution of the local updates to other clients. While the effort is negligible in the considered cross-silo scenario of a few collaborating computing centers, this overhead might become more relevant when applying CrowdGuard to other scenarios with large numbers of participants. In Sect. VI-B, we discussed the option to consider a subset of all clients for validation to the cost of stronger security assumptions required. Analogously, this can also be used to reduce the computation overhead.

VII. Related Work

In the recent past, a large number of backdoor attacks and defenses have been proposed. In the following, we discuss the approaches that are most relevant to this work and categorize them into the following types: DF approaches that aim to detect backdoored models (Sect. VII-A) and IR approaches that mitigate the backdoors without identifying the poisoned models (Sect. VII-B). Afterwards, we will investigate privacy attacks and defenses (Sect. VII-C).

A. Filtering Approaches

Auror [81] clusters selected parameters of the model updates on the server using k-means. In comparison, CrowdGuard considers the outputs of all neurons. Additionally, Auror is vulnerable to multi-backdoor attacks where different clients inject different backdoors [65], like in our combined attack scenarios. The significance-test-based algorithm of CrowdGuard handles such attacks by iteratively pruning the backdoored models.

FoolsGold [28] assumes all clients to be non-IID. For its analysis, it sums up all model updates that each client submitted to create a client-specific update history, before using the Cosine to compare the update histories of different clients. However, it can be circumvented by adaptive attacks [7] and fails to handle IID scenarios. Further, the update history allows an adversary to gain trust by behaving benign for several rounds before performing its attack. Flame [65] combines an outlier-detection-based approach with clipping and noising. However, the noising reduces the performance of the model, while the outlier detection fails in non-IID scenarios. In comparison, CrowdGuard does not affect the performance of the models and handles IID and advanced non-IID scenarios even with disjoint data.

DeepSight [73] uses different techniques to extract fingerprints of the training data from the models' parameters and

predictions to distinguish benign and backdoored models. Its classification relies on the assumption that poisoned models were trained on fewer labels as benign models. However, e.g., if the benign clients are trained only on a single label (cf. experiments for q=0.0 in Sect. V), this assumption does not hold, while CrowdGuard can also in this corner case effectively distinguish benign and backdoored updates.

Zhao *et al.* [96] is the closest to our work, as it also inspects the local models on the client side. The detection mechanism is based on the local models performance (MA) on the clients' local data, which means the output of the last layer only. To protect the privacy of the local models, Zhao *et al.* rely on applying differential privacy (DP) on the local models. In comparison, CrowdGuard analyses the deep layer outputs and also detects sophisticated backdoors in models that do not affect the predictions for benign samples, therefore maintaining the MA, which is beneficial, since stealthy backdoor attacks do not affect MA (O2 in Sect. III-B). Further, performing the client-side analysis in TEEs guarantees the privacy of the local models but does not affect the analysis results. In comparison, applying DP, i.e., adding noise, changes the models and thus affects the results. Also, it is challenging to determine suitable parameters for DP, as too low parameters do not protect privacy while too strong parameters significantly affect the models' predictions and thus also the analysis results. Additionally, the clients have to report the number of samples for each label to the server to enable the server to choose suitable validation clients for each model [96]. The filtering decision of Zhao *et al.* is based on empirical thresholds instead of statistical tests as utilized in our approach. Hence, CrowdGuard is independent of the dataset and model which results in a better real-world applicability.

FLARE utilizes a server-side dataset and applies KNN on a single layer's plain outputs [89]. However, assuming the presence of a server-side dataset is not practical [73]. In addition, focusing the analysis on a single layer allows the adversary to bypass the defense by fixing this single layer and hiding in others. Thus, CrowdGuard is the first approach that uses all layers' outputs to identify poisoned models without triggered samples.

B. Mitigation Approaches

Other approaches try to mitigate the backdoor without identifying the poisoned models. Yin *et al.* [94] proposed two approaches: One uses the parameters' median as a rule for aggregation, while the other one removes extreme parameter values and aggregates the remaining ones. Krum [10] selects a single model as an aggregated model that minimizes the distance to a fraction of other models. However, the approaches of Yin *et al.* [94] and Blanchard *et al.* [10] do not work well in non-IID scenarios, preventing benign-but-outlier models from being included in the aggregation. Naseri *et al.* [62] propose using Differential Privacy (DP) for mitigating backdoors. However, besides the drawback that this strategy always reduces the model's performance, the DP level needs to be chosen manually. Here, a too high value makes the model unusable while a too low value is ineffective. In comparison, CrowdGuard works without any dataset-specific parameters and relies on the significance level (p-value) of the statistical tests instead. Chen *et al.* look on models' behavior during prediction and detects if input data triggering a backdoor are used [16]. For triggered samples, the predictions differ obviously, making detection straightforward. By leveraging HLBIM, CrowdGuard's validation algorithm allows detecting poisoned models without triggered data, even if the DLOs differ only marginally. BaFFLe [6] aggregates all local models and analyzes the final layer's output of the aggregated model via client feedback. Placing the backdoor detection after the aggregation allows the usage of secure-aggregation schemes. Therefore, BaFFLe relies for privacy protection on the security of the chosen secure-aggregation mechanism and the anonymity of the aggregated model. In contrast, CrowdGuard inspects every layer of the individual local models on the client side. Therefore, BaFFLe's backdoor identification is based on a postulate that backdoors affect the predicted class for regular data which is not practical (see O2 in Sect. III-B). Further, BaFFLe cannot detect attacks in early rounds but needs multiple benign rounds for building a benign history, requires several empirically determined thresholds, and discards the whole training round if an attack is detected. In comparison, CrowdGuard is round independent, uses only statistical thresholds based on probabilities that are independent of dataset and model, and allows training a model even in the presence of adversaries by filtering the poisoned updates.

In addition, all IR approaches have the disadvantage that the malicious clients cannot be identified and, hence, permanently excluded, but will permanently try to inject the backdoor, requiring the respective defense to always perfectly mitigate the poisoned models.

C. Privacy Attacks and Defenses

Privacy Attacks: There are several attacks against ML models that are capable of leaking private information such as membership inference attacks [36], [82], property inference attacks [29], and label inference attacks [95]. Additionally, inference methods that reconstruct the whole input have been developed [76] not only for centralized ML processes but also in the area of FL [90]. The TEE-based architecture of CrowdGuard prevents inference attacks on the local models before the aggregation anonymizes the individual clients' contributions, while the aggregation of many models impedes inference attacks on the aggregated model [63]. Thus, not only attacks are detected, but the real-world applicability of FL is pushed.

Secure Aggregation Techniques: Various approaches have been proposed to prevent honest-but-curious [26] or fully malicious servers [11], [60], [35] from accessing the local model updates. For example, Bonawitz *et al.* [11] use a secret-sharing protocol to allow the clients the calculation of noise that will cancel out during aggregation. However, this approach is not compatible with state-of-the-art backdoor defenses. Fereidooni *et al.* [26] use secure multi-party computation. However, these approaches create significant overhead for the clients and server. In the past, different approaches have been proposed using TEEs for the aggregation step. In PPFL [60], the whole FL process (training and aggregation) is performed inside a TEE. Hashemi *et al.* [35] implemented Krum [10] on SGX. In comparison to both, cryptograpy-based and TEE-based secure aggregation, CrowdGuard not only implements secure aggregation inside a TEE. Instead, CrowdGuard also

provides an architecture to securely leverage clients' data for backdoor detection, without taking any privacy risk for local models or datasets.

VIII. Conclusion

Privacy of sensitive data and defenses against poisoning attacks are central security considerations when it comes to Federated Learning (FL). To satisfy these needs, we propose CrowdGuard, a model filtering defense against targeted poisoning attacks that introduces a client feedback loop leveraging the clients' local data for model assessment. In contrast to existing approaches, CrowdGuard does not only rely on the vector metrics or models' accuracies but analyzes changes in the behavior of the deep layers' neurons to identify backdoor behavior. This enables CrowdGuard to identify poisoned models independent of the clients' data distribution or the attack strategy.

CrowdGuard has three core components: 1) A novel TEE-based architecture that allows using clients' data for the model validation without creating new privacy-attack vectors. 2) A significance-based backdoor detection algorithm that executes statistical tests operating on HLBIM, a novel metric based on the deep layer outputs of local models allowing to identify adversarial models. 3) A stacked clustering scheme, which compensates rogue votes of adversarial clients during the feedback loop. Thereby, our proposed architecture preserves the privacy, integrity, and confidentiality of local models and consequently client data by leveraging secure environments.

We evaluate our approach in various FL settings and show the independence of those factors. Additionally, CrowdGuard does not reduce the FL performance and is not circumventable by adaptive adversaries that are aware of the defense, making it applicable in real-world scenarios.

Acknowledgment

This research received funding from Intel through the Private AI Collaborate Research Institute (https://www.private-ai.org/), as well as from the OpenS3 Lab and the Hessian Ministry of Interior and Sport as part of the F-LION project, following the funding guidelines for cyber security research.

References

[1] The federated tumor segmentation (FeTS) initiative. https://www.med.upenn.edu/cbica/fets/#FeTSCollaboratingSites6. Accessed: 2023-04-02.

[2] Health Insurance Portability and Accountability Act, 1996. https://www.govinfo.gov/content/pkg/PLAW-104publ191/pdf/PLAW-104publ191.pdf.

[3] California Consumer Privacy Act, 2018. https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180SB1121.

[4] General Data Protection Regulation, 2018. https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[5] Pytorch, 2022. https://pytorch.org.

[6] Sebastien Andreina, Giorgia Azzurra Marson, Helen Möllering, and Ghassan Karame. BaFFLe: Backdoor Detection via Feedback-based Federated Learning. In *ICDCS*, 2021.

[7] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How To Backdoor Federated Learning. In *AISTATS*, 2020.

[8] Mauro Barni, Kassem Kallas, and Benedetta Tondi. A new backdoor attack in cnns by training set corruption without label poisoning. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 101–105. IEEE, 2019.

[9] Gilles Barthe, Sandrine Blazy, Benjamin Grégoire, Rémi Hutin, Vincent Laporte, David Pichardie, and Alix Trieu. Formal verification of a constant-time preserving c compiler. *Proceedings of the ACM on Programming Languages*, 2020.

[10] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In *NIPS*, 2017.

[11] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *CCS*, 2017.

[12] Ferdinand Brasser, Srdjan Capkun, Alexandra Dmitrienko, Tommaso Frassetto, Kari Kostiainen, and Ahmad-Reza Sadeghi. Dr. sgx: Automated and adjustable side-channel protection for sgx using data location randomization. In *CCS*, 2019.

[13] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. In *USENIX Workshop on Offensive Technologies (WOOT 17)*. USENIX Association, 2017.

[14] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. In *NDSS*, 2021.

[15] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Provably secure federated learning against malicious clients. In *AAAI Conference on Artificial Intelligence*, 2021.

[16] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[17] Fei Chen, Mi Luo, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning with fast convergence and efficient communication. In *arXiv preprint arXiv:1802.07876*, 2018.

[18] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. In *arXiv preprint arXiv:1712.05526*, 2017.

[19] Victor Costan and Srinivas Devadas. Intel sgx explained. In *Cryptology ePrint Archive*, 2016.

[20] Lesly-Ann Daniel, Sébastien Bardin, and Tamara Rezk. Binsec/rel: Efficient relational symbolic execution for constant-time at binary-level. In *IEEE S&P*. IEEE, 2020.

[21] Li Deng. The mnist database of handwritten digit images for machine learning research. In *IEEE Signal Processing Magazine*, volume 29, pages 141–142. IEEE, 2012.

[22] Jean-Antoine Désidéri. Multiple-gradient descent algorithm (mgda) for multiobjective optimization. *Comptes Rendus Mathematique*, 350(5):313–318, 2012.

[23] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. 1996.

[24] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *USENIX Security*, 2020.

[25] Hossein Fereidooni, Alexandra Dmitrienko, Phillip Rieger, Markus Miettinen, Ahmad-Reza Sadeghi, and Felix Madlener. Fedcri: Federated mobile cyber-risk intelligence. In *NDSS*, 2022.

[26] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, et al. Safelearn: Secure aggregation for private federated learning. In *IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021.

[27] Patrick Foley, Micah J Sheller, Brandon Edwards, Sarthak Pati, Walter Riviera, Mansi Sharma, Prakash Narayana Moorthy, Shi-han Wang, Jason Martin, Parsa Mirhaji, Prashant Shah, and Spyridon Bakas. Openfl: the open federated learning library. *Physics in Medicine & Biology*, 2022.

[28] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. The limitations of federated learning in sybil settings. In *RAID*, 2020.

[29] Karan Ganju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In *CCS*, 2018.

[30] Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyoungshick Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review. In *arXiv preprint arXiv:2007.10760*, 2020.

[31] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *arXiv preprint arXiv:1708.06733*, 2017.

[32] Rachid Guerraoui, Sébastien Rouault, et al. The hidden vulnerability of distributed learning in byzantium. In *International Conference on Machine Learning*, pages 3521–3530. PMLR, 2018.

[33] Gozde N Gunesli, Mohsin Bilal, Shan E Ahmed Raza, and Nasir M Rajpoot. Feddropoutavg: Generalizable federated learning for histopathology image classification. In *arXiv preprint arXiv:2111.13230*, 2021.

[34] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. In *arXiv preprint arXiv:1811.03604*, 2018.

[35] Hanieh Hashemi, Yongqin Wang, Chuan Guo, and Murali Annavaram. Byzantine-robust and privacy-preserving framework for fedml. In *ICLR Workshops*, 2021.

[36] Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. Logan: Membership inference attacks against generative models. In *Privacy Enhancing Technologies*, 2019.

[37] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. In *arXiv preprint arXiv:1909.06335*, 2019.

[38] Wei Huang, Shengjie Xu, Yueqiang Cheng, and David Lie. Aion attacks: Manipulating software timers in trusted execution environment. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 173–193. Springer, 2021.

[39] David Kaplan, Jeremy Powell, and Tom Woller. Amd memory encryption. In *White paper*, 2016.

[40] Youssef Khazbak, Tianxiang Tan, and Guohong Cao. Mlguard: Mitigating poisoning attacks in privacy preserving distributed collaborative learning. In *International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2020.

[41] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. In *arXiv preprint arXiv:1610.02527*, 2016.

[42] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[43] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Citeseer, 2009.

[44] Kavita Kumari, Phillip Rieger, Hossein Fereidooni, Murtuza Jadliwala, and Ahmad-Reza Sadeghi. Baybfed: Bayesian backdoor defense for federated learning. In *IEEE S&P*. IEEE Computer Society, 2023.

[45] Huimin Li, Phillip Rieger, Shaza Zeitouni, Stjepan Picek, and Ahmad-Reza Sadeghi. Flairs: Fpga-accelerated inference-resistant & secure federated learning. *arXiv preprint arXiv:2308.00553*, 2023.

[46] Liping Li, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling. Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *AAAI Conference on Artificial Intelligence*, 2019.

[47] Mingyu Li, Yubin Xia, and Haibo Chen. Memory optimization system for sgxv2 trusted execution environment. *International Journal of Software & Informatics*, 12(3), 2022.

[48] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2019.

[49] Xiaoxiao Li, Meirui Jiang, Xiaofei Zhang, Michael Kamp, and Qi Dou. Fedbn: Federated learning on non-iid features via local batch normalization. In *arXiv preprint arXiv:2102.07623*, 2021.

[50] Tjen-Sien Lim and Wei-Yin Loh. A comparison of tests of equality of variances. *Computational Statistics & Data Analysis*, 22(3):287–301, 1996.

[51] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. Composite backdoor attack for deep neural network by mixing existing benign features. In *CCS*, New York, NY, USA, 2020. Association for Computing Machinery.

[52] Pengrui Liu, Xiangrui Xu, and Wei Wang. Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives. *Cybersecurity*, 5(1), 2022.

[53] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and X. Zhang. Trojaning attack on neural networks. In *NDSS*, 2018.

[54] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *European Conference on Computer Vision*, pages 182–199. Springer, 2020.

[55] Edward H Livingston. Who was student and why do we care so much about his t-test? 1. *Journal of Surgical Research*, 118(1):58–65, 2004.

[56] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*, 2017.

[57] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative Machine Learning without Centralized Training Data. Google AI, 2017.

[58] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning Differentially Private Language Models Without Losing Accuracy. In *ICLR*, 2018.

[59] Thomas Minka. Estimating a dirichlet distribution. 01 2003.

[60] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. Ppfl: privacy-preserving federated learning with trusted execution environments. In *Annual International Conference on Mobile Systems, Applications, and Services*, 2021.

[61] Luis Muñoz-González, Kenneth T. Co, and Emil C. Lupu. Byzantine-Robust Federated Machine Learning through Adaptive Model Averaging. In *arXiv preprint:1909.05125*, 2019.

[62] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. Local and central differential privacy for robustness and privacy in federated learning. In *NDSS*, 2022.

[63] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE S&P*. IEEE, 2019.

[64] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N. Asokan, and Ahmad-Reza Sadeghi. DÏoT: A Federated Self-learning Anomaly Detection System for IoT. In *ICDCS*, 2019.

[65] Thien Duc Nguyen, Phillip Rieger, Huili Chen, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni. FLAME: taming backdoors in federated learning. In *USENIX Security*, 2022.

[66] Thien Duc Nguyen, Phillip Rieger, Markus Miettinen, and Ahmad-Reza Sadeghi. Poisoning Attacks on Federated Learning-Based IoT Intrusion Detection System. In *Workshop on Decentralized IoT Systems and Security*, 2020.

[67] Frank Nielsen. *Hierarchical Clustering*, pages 195–211. 02 2016.

[68] Nvidia. Nvidia h100 tensor core gpu architecture. In *White paper*, 2022.

[69] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Comput. Surv.*, 51(6), jan 2019.

[70] Friedrich Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, 1994.

[71] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. Knock knock, who's there? membership inference on aggregate location data. In *NDSS*, 2018.

[72] John A Rice. *Mathematical statistics and data analysis*. Cengage Learning, 2006.

[73] Phillip Rieger, Thien Duc Nguyen, Markus Miettinen, and Ahmad-Reza Sadeghi. Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection. In *NDSS*, 2022.

[74] Holger R Roth, Ken Chang, Praveer Singh, Nir Neumark, Wenqi Li, Vikash Gupta, Sharut Gupta, Liangqiong Qu, Alvin Ihsani, Bernardo C Bizzo, et al. Federated learning for breast density classification: A real-world implementation. In *Domain Adaptation and Representation*

Transfer, and Distributed and Collaborative Learning, pages 181–191. Springer International Publishing, 2020.

[75] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J Doug Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *ACM SIGCOMM Conference on Internet Measurement*, 2009.

[76] Ahmed Salem, Apratim Bhattacharya, Michael Backes, Mario Fritz, and Yang Zhang. Updates-leak: Data set inference and reconstruction attacks in online learning. In *USENIX Security*, 2020.

[77] Fan Sang, Ming-Wei Shih, Sangho Lee, Xiaokuan Zhang, Michael Steiner, Mona Vij, and Taesoo Kim. Pridwen: Universally hardening sgx programs via load-time synthesis. In *USENIX Security*, 2022.

[78] Micah Sheller, Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Federated Learning for Medical Imaging. In *Intel AI*, 2018.

[79] Micah Sheller, Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Multi-Institutional Deep Learning Modeling Without Sharing Patient Data: A Feasibility Study on Brain Tumor Segmentation. In *Brain Lesion Workshop*, 2018.

[80] Micah J. Sheller, Brandon Edwards, G. Anthony Reina, Jason Martin, Sarthak Pati, Aikaterini Kotrotsou, Mikhail Milchenko, Weilin Xu, Daniel Marcus, Rivka R. Colen, and Spyridon Bakas. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. In *Scientific Reports*, volume 10, page 12598, 2020.

[81] Shiqi Shen, Shruti Tople, and Prateek Saxena. Auror: Defending Against Poisoning Attacks in Collaborative Deep Learning Systems. In *ACSAC*, 2016.

[82] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE S&P*. IEEE, 2017.

[83] Chia-Che Tsai, Donald E Porter, and Mona Vij. {Graphene-SGX}: A practical library {OS} for unmodified applications on {SGX}. In *USENIX Annual Technical Conference (USENIX ATC)*, 2017.

[84] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. Cacheout: Leaking data on intel cpus via cache evictions. In *IEEE S&P*. IEEE, 2021.

[85] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on gpus. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 681–696, 2018.

[86] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. In *NIPS*, volume 33, 2020.

[87] Jinwen Wang, Yueqiang Cheng, Qi Li, and Yong Jiang. Interface-based side channel attack against intel sgx. In *arXiv preprint arXiv:1811.05378*, 2018.

[88] Lixu Wang, Shichao Xu, Xiao Wang, and Qi Zhu. Eavesdrop the Composition Proportion of Training Labels in Federated Learning. In *arXiv preprint:1910.06044*, 2019.

[89] Ning Wang, Yang Xiao, Yimin Chen, Yang Hu, Wenjing Lou, and Y Thomas Hou. Flare: defending federated learning against model poisoning attacks via latent space representations. In *Asia Conference on Computer and Communications Security*, 2022.

[90] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019.

[91] Zhaoxian Wu, Qing Ling, Tianyi Chen, and Georgios B. Giannakis. Federated variance-reduced stochastic gradient descent with robustness to byzantine attacks. In *IEEE Transactions on Signal Processing*, volume 68, pages 4583–4596, 2020.

[92] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2020.

[93] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), jan 2019.

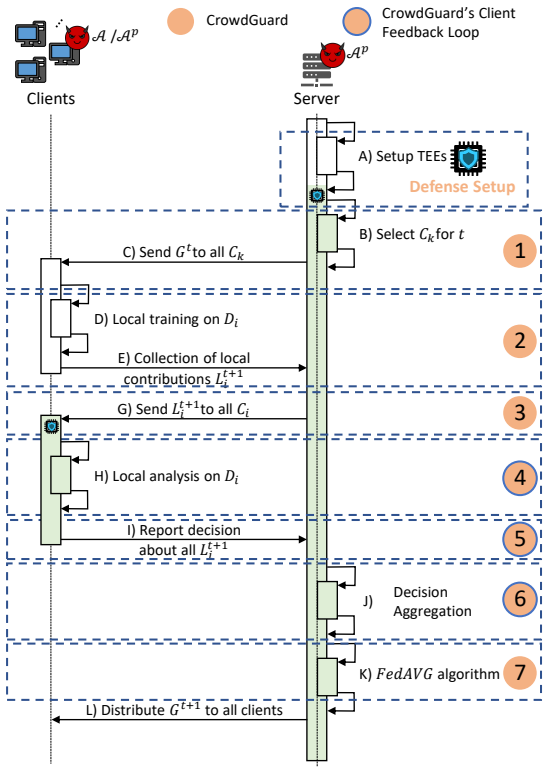


Fig. 5: Execution sequence of CrowdGuard. Green indicates the utilization of secure environments.

[94] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, pages 5650–5659. PMLR, 2018.

[95] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. In *arXiv preprint arXiv:2001.02610*, 2020.

[96] Lingchen Zhao, Shengshan Hu, Qian Wang, Jianlin Jiang, Chao Shen, Xiangyang Luo, and Pengfei Hu. Shielding collaborative learning: Mitigating poisoning attacks through client-side detection. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2029–2041, 2020.

[97] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. Federated learning on non-iid data: A survey. In *Neurocomputing*, volume 465, pages 371–390. Elsevier, 2021.

[98] Jianping Zhu, Rui Hou, XiaoFeng Wang, Wenhao Wang, Jiangfeng Cao, Lutan Zhao, Fengkai Yuan, Peinan Li, Zhongpu Wang, Boyan Zhao, Lixin Zhang, and Dan Meng. Enabling privacy-preserving, compute- and data-intensive computing using heterogeneous trusted execution environment, 2019.

APPENDIX

Fig. 5 depicts a detailed overview of the FL setup with CrowdGuard activated as a defense mechanism against targeted poisoning attacks.

Alg. 4 shows the algorithm that the individual clients execute during the validation to analyze the HLBIM values and determine, whether there is still a significance for the presence of poisoned models in the calculated PCA values, such that CrowdGuard needs to perform another pruning iteration.

It is also a valid proposition to consider leveraging multiple PC dimensions instead of solely relying on PC dimension one. However, our experiments have consistently demonstrated that the first PC dimension encompasses over 65% (mostly

exceeding 80%) of the explained variances obtained through PCA. This measure signifies the significance and efficacy of this particular dimension in effectively segregating the data points. Incorporating the second dimension merely contributes to a marginal increase in the explained variances, which is why we opt to prioritize the utilization of the first dimension. This approach not only ensures computational efficiency but also maintains an intuitive framework.

An additional factor that influences our decision is the internal methods employed by CrowdGuard. Specifically, the statistical tests implemented within CrowdGuard are specifically designed to handle one-dimensional data. Employing multiple dimensions would necessitate the implementation of alternative algorithms, such as clustering methods. However, the statistical tests constitute the core of CrowdGuard and are instrumental in yielding the positive effects associated with this defense mechanism.

Algorithm 4 Significance Test on PC Values

```
Input:
pc_dim1_values,                                   ▷ A list of values
Output:
significant          ▷ Indicator if the values are contain abnormalities
                                                  ▷ Generate distributions
median ← MEDIAN(pc_dim1_values)
upper ← {}
lower ← {}
for value in pc_dim1_values do
    distribution_value ← value − median
    if value >= 0 then
        upper.append(distribution_value)
    else
        lower.append(abs(distribution_value))
    end if
end for
                                                  ▷ Significance tests
mean_significant ← T-TEST(upper, lower)
var_significant ← F-TEST(upper, lower)
dist_significant ← D-TEST(upper, lower)
outlier_quartil_significant ← OUTLIER_BOXPLOT(pc_dim1_values)
outlier_sigma_significant ← OUTLIER_3σ(pc_dim1_values)
                                                  ▷ Aggregate result
significant ← mean_significant OR var_significant OR dist_significant
OR outlier_quartil_significant OR outlier_sigma_significant
```

Research knows about various kinds of triggers for different scenarios, i.e. [51], [18], but we will only explain few of them, which are also used in our experiments:

- **Pixel Backdoor**: This is a backdoor in the domain of image classification, where a pixel pattern is placed on the benign input image [7], [31], [53], as visualized in Fig. 1a and the label is changed to the desired $T_{\mathcal{A}}$. In another injection strategy called *Distributed Backdoor*, this trigger is distributed between multiple adversarial clients. Each client incorporates a fraction of the pattern into their local model. The final trigger is a combination of all the fractions [92].
- **Label Swap**: All samples of one label are swapped to $T_{\mathcal{A}}$. To create a poisoned dataset $\mathcal{D}_i^{\mathcal{A}}$, only changes regarding the label mapping are mandatory in $\mathcal{D}_i$.
- **Semantic Backdoor**: In this case, the input data contain a specific characteristic within the benign image, that should trigger a swap to $T_{\mathcal{A}}$. Examples regarding the CIFAR-10 [43] dataset are the mapping of cars in front of a striped background (cf. Fig. 1b) to $T_{\mathcal{A}}$, but leave all other car samples like Fig. 1c in its benign states [7].

Default Configurations	
Parameter	Default Value
Dataset	CIFAR-10
Clients n	$\lvert C_k\rvert = \lvert C_i\rvert = 20$
Epochs	10
Samples per client	2560
Batch Size	64
Backdoor	Semantic Backdoor
IID rate q	0
Poison Data Rate (PDR)	0.1
Starting round t	1000
Adaptive adversary rate[18] α	0.7
Poison Model Rate (PMR)	0.45 (= 9/20)
Benign Learning Rate	0.01
Malicious Learning Rate	0.01

TABLE IV: Listing of the default FL setup configurations.

In the past, different side-channel attacks have been proposed that extract data from TEEs [13], [38], [87]. Different targets for such attacks in CrowdGuard are possible. The direct target would be using side-channel attacks to first extract the local models of other clients from an enclave and then perform a model inference attack [29], [36], [52], [63], [71], [76], [82], [88], [95] on the extracted models. However, existing inference attacks have a low bandwidth of less than 100 bytes/s [84], which is negligible compared to the size of a DNN model.

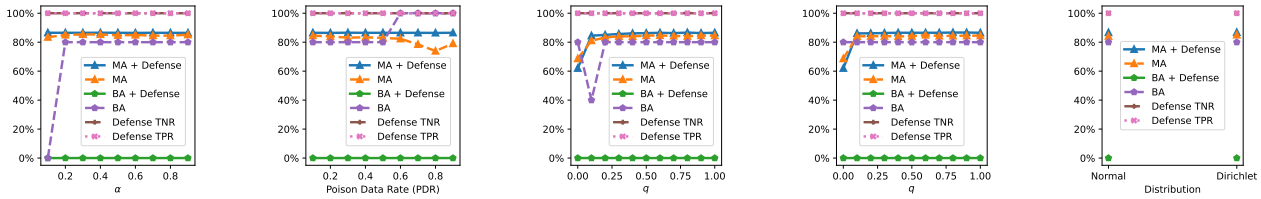
Another option would be using such side-channel attacks to extract the cryptographic keys from the enclave and use this key to fake a TEE, thus to break the TEE completely. Also, an attacker could try to extract the keys that are used for the encrypted communication, i.e., the TLS session key, use the extracted key to eavesdrop the local models during their transmission and then run a model inference attack against the eavesdropped models. However, while attacks are proposed, defenses against such attacks are also frequently developed. Examples against such side-channel attacks include the usage of constant-time encryption algorithms [9], [20] or techniques that randomized the data locations inside the memory [12], [77]. Thus, we consider such attacks to be out of the scope of this paper and assume TEEs to be trusted.

The probability of violating the majority assumption, when a subset of all clients is randomly selected for validation, follows a hypergeometric distribution [72]. Fig. 7 shows the probability that more than 50% of the selected validation clients are malicious for different Poisoned Model Rates (PMRs) and different numbers of validation clients (for overall 1000 clients). As it can be seen, the probability for small PMR values becomes negligible already for less than 50 validation clients.

In Tab. IV, the default parameter configurations for our experiments are depicted.

Besides the accuracy on the benign main task (Main Task Accuracy, MA) and the backdoor task (Backdoor Accuracy, BA), we also evaluate the filtering capabilities of CrowdGuard by measuring the True Positive Rate (TPR) and True Negative Rate (TNR). For this purpose, we consider a benign model that is correctly recognized by CrowdGuard to be a True Negative (TN), a poisoned model that is correctly identified as True Positive (TP) and analogously for False Positives (FP) and

[18] Adaptive adversary from Bagdasaryan *et al.* [7].



(a) α (b) PDR (c) 1-class non-IID (d) 2-class non-IID (e) Distributions

Fig. 6: Influence of parameters.

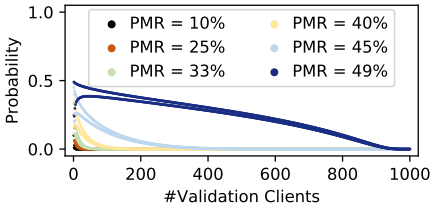
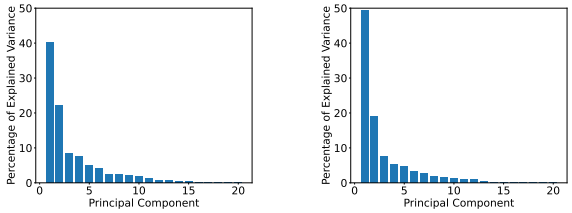


Fig. 7: Probability for more than 50% of adversaries being selected as validators out of 1000 clients for different PMRs.



(a) Absolute (b) Relative

Fig. 8: Comparison of the expected variance of the PCA with absolute or relative distances used in the routine of CrowdGuard. The figures are based on the Cosine distances in the first pruning round.

Scenario	Majority		K-Means		Agglomerative		DBSCAN		CrowdGuard	
	TPR	TNR	TPR	TNR	TPR	TNR	TPR	TNR	TPR	TNR
Default	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
All Benign + 1 FN	88.9	100.0	88.9	100.0	88.9	100.0	100.0	100.0	100.0	100.0
Default + 2 FP	100.0	100.0	100.0	100.0	100.0	100.0	0.0	100.0	100.0	100.0
All Benign + 2 FN	77.8	100.0	77.8	100.0	77.8	100.0	0.0	100.0	100.0	100.0
Malicious Split	100.0	100.0	0.0	100.0	0.0	100.0	100.0	100.0	100.0	100.0

TABLE V: Experimental result for the comparison of different aggregation rules for combining individual votes.

False Negatives (FN). The TPR is then defined as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4}$$

The TNR is calculated as:

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{5}$$

As discussed in Sect. III-B, we assume that the malicious clients can submit arbitrary votes by providing corresponding data to their enclave.

The stacked clustering ensures the integrity of the filtering, even if $\mathcal{A}$ can manipulate the votes that are reported by the enclaves of the malicious clients (cf. Alg. 3). Tab. V shows the results of our ablation study for the vote aggregation of CrowdGuard. We compare the stacked-clustering with different alternatives, in particular, majority voting, where a model is rejected if a majority of clients votes for its rejection, and K-Means, where the votes of the individual clients are clustered using K-Means and a model is accepted if at least one client in the majority cluster votes for acceptance. In addition, we evaluated the individual components of the stacked clustering (Agglomerative and DBSCAN) separately. We evaluate these aggregation rules for the votes that we observed for our default setting (Default). Further, we consider two scenarios, where all malicious clients vote for accepting all models but one/two benign clients do not detect one poisoned model (All Benign + 1 FN and All Benign + 2 FN). Further, we evaluate a scenario where all clients vote as observed in our default setting but two clients consider a single benign model to be poisoned (Default + 2 FP). In another setting, the adversary splits the clients into two groups: One group votes for accepting all models and one group for rejecting all models (Malicious Split). As Tab. V shows, only the stacked clustering of CrowdGuard always detects all malicious models (TPR=100%), while the other aggregation rules miss some of the models, or, in the case of K-Means, even accept all poisoned models.

To analyze the impact of using the relative distance in the HLBIM metric, we conducted an additional experiment for an adapted version of HLBIM, that utilizes absolute distances instead of relative versions. However, we observed that this adapted metric provides less insight into the models' behavior and it is more challenging to use these values to distinguish between benign and poisoned models.

The reason for this is that utilizing relative distances allows us to consider the relative magnitude of updates. When a given parameter value is relatively small compared to another parameter that undergoes the same update value, the relative update becomes more relevant. After executing the PCA, this is also visible in the capability of the first Principal Component (PC) dimension to separate the data points, as measured by the explained variance. By leveraging the relative distances, we were able to enhance this capability by an average of 10%, as can be seen in Fig. 8. This improvement in separation capability resulted in better overall performance, particularly in edge cases.

Thus, using relative distances proved to be more meaningful and beneficial for the HLBIM, allowing for improved detection and differentiation of poisoned models from benign values.

In Fig. 6 We depict graphs illustrating the influence of parameters in Fig. 6. As it can be seen, the defense is independent of α, the PDR, and the non-IID scenario and achieves 100% True-Positive-Rate (TPR) as well as True-Negative-Rate (TNR). The Main Task Accuracy (MA) is higher if the defense is activated, so we do not decrease the benign FL performance.