
Sagar Pathak

Follow this and additional works at: https://digitalcommons.memphis.edu/etd

Recommended Citation

This Thesis is brought to you for free and open access by University of Memphis Digital Commons. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of University of Memphis Digital Commons. For more information, please contact khggerty@memphis.edu.
FORTIFYING FEDERATED LEARNING: A COMPREHENSIVE ANALYSIS AND NOVEL SOLUTIONS FOR PRIVACY AND SECURITY ISSUES IN FEDERATED LEARNING

by

Sagar Pathak

A Thesis
Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science

Major: Computer Science

The University of Memphis
August 2024
Abstract

Data exhibits a natural distribution, and effective machine learning models rely on substantial datasets. However, much data remains inaccessible due to privacy and security risks in traditional centralized settings, which require data collection on a central server. Federated Learning (FL) addresses this by moving computation to the data source instead of the server. Despite this, FL faces challenges from data and model poisoning attacks due to its distributed nature. Verifying the authenticity of clients in FL is difficult. Proposed solutions include statistical analysis of client updates, hardware-based isolation, Differential Privacy (DP), and Homomorphic Encryption (HE). However, these solutions have limitations and significant trade-offs, such as the privacy-utility trade-off. This research proposes a novel approach to fortify the FL environment using Zero-Trust (ZT) inspired continuous verification of client updates for model poisoning attacks and filter ensembles for data poisoning attacks. Our experiment demonstrates improved results against these attacks.

Keywords: Federated Learning, Differential Privacy (DP), Homomorphic Encryption (HE), Zero-Trust (ZT), Privacy and Security, Poisoning Attacks
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>Abbreviations</td>
<td>viii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Research Motivation</td>
<td>2</td>
</tr>
<tr>
<td>Research Gaps</td>
<td>2</td>
</tr>
<tr>
<td>Research Challenges</td>
<td>3</td>
</tr>
<tr>
<td>2 Background Study</td>
<td>5</td>
</tr>
<tr>
<td>Key components of Federated Learning</td>
<td>5</td>
</tr>
<tr>
<td>Central Aggregation Server (CAS)</td>
<td>6</td>
</tr>
<tr>
<td>Clients (Nodes)</td>
<td>6</td>
</tr>
<tr>
<td>Global Model</td>
<td>6</td>
</tr>
<tr>
<td>FL Algorithms</td>
<td>6</td>
</tr>
<tr>
<td>Types of Federated Learning</td>
<td>8</td>
</tr>
<tr>
<td>Horizontal Federated Learning</td>
<td>8</td>
</tr>
<tr>
<td>Vertical Federated Learning</td>
<td>8</td>
</tr>
<tr>
<td>Federated Transfer Learning</td>
<td>8</td>
</tr>
<tr>
<td>Core Principles of Federated Learning</td>
<td>9</td>
</tr>
<tr>
<td>Collaborative Learning</td>
<td>9</td>
</tr>
<tr>
<td>Privacy and Security</td>
<td>10</td>
</tr>
<tr>
<td>Decentralized Data</td>
<td>10</td>
</tr>
<tr>
<td>Challenges in Federated Learning</td>
<td>11</td>
</tr>
<tr>
<td>Scalability and Communication Overheads</td>
<td>11</td>
</tr>
<tr>
<td>Data Heterogeneity</td>
<td>11</td>
</tr>
<tr>
<td>Data Privacy and Security</td>
<td>12</td>
</tr>
<tr>
<td>Convergence Problem</td>
<td>12</td>
</tr>
<tr>
<td>3 Related Work</td>
<td>13</td>
</tr>
<tr>
<td>Statistical Anomaly Detection Approaches</td>
<td>13</td>
</tr>
<tr>
<td>Secure Aggregation Approaches</td>
<td>16</td>
</tr>
<tr>
<td>Hardware Based Approaches</td>
<td>18</td>
</tr>
<tr>
<td>Filter Ensemble Based Approaches</td>
<td>19</td>
</tr>
<tr>
<td>4 Data Privacy and Security</td>
<td>20</td>
</tr>
<tr>
<td>General Data Protection and Regulation (GDPR)</td>
<td>20</td>
</tr>
<tr>
<td>California Consumer Privacy Act (CCPA)</td>
<td>20</td>
</tr>
<tr>
<td>Existing Privacy and Security Attacks</td>
<td>21</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Client Modalities Sample Table</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>Random forest model used in the filter ensemble for the experiment</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Experiment details against data poisoning attack</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>Experiment details against model poisoning attack</td>
<td>44</td>
</tr>
<tr>
<td>5</td>
<td>Experiment details against data and model poisoning attack</td>
<td>45</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example of next-word prediction NLP task using FL</td>
</tr>
<tr>
<td>2</td>
<td>Comparison between classical and FL Approach</td>
</tr>
<tr>
<td>3</td>
<td>Collaborative learning for the next-word predictions on mobile phones using federated learning</td>
</tr>
<tr>
<td>4</td>
<td>Concept drift in client due to data heterogeneity across time</td>
</tr>
<tr>
<td>5</td>
<td>Example of membership inference attack</td>
</tr>
<tr>
<td>6</td>
<td>Illustration of poisoned data and model in the FL environment</td>
</tr>
<tr>
<td>7</td>
<td>Venn diagram showing the relationship among the poisoning attacks</td>
</tr>
<tr>
<td>8</td>
<td>Fake images generated by GAN using the website thispersondoesnotexist.com</td>
</tr>
<tr>
<td>9</td>
<td>Illustration of CDP and LDP in FL</td>
</tr>
<tr>
<td>10</td>
<td>Symmetric key encryption (in client) and decryption (in server) using the value of the modality ID</td>
</tr>
<tr>
<td>11</td>
<td>Overall process of the proposed FL process for continuous verification using client modalities</td>
</tr>
<tr>
<td>12</td>
<td>Use of filter ensemble in FL framework</td>
</tr>
<tr>
<td>13</td>
<td>FL Process Orchestration in customized Flower Framework</td>
</tr>
<tr>
<td>14</td>
<td>Sample images from MNIST dataset</td>
</tr>
<tr>
<td>15</td>
<td>Illustration of the Neural Network used for MNIST handwritten digits recognition</td>
</tr>
<tr>
<td>16</td>
<td>Results showing the effectiveness of the filter ensemble against Data Poisoning Attack</td>
</tr>
<tr>
<td>17</td>
<td>Results showing the effectiveness of the ZeroFedAvg against Model Poisoning Attack</td>
</tr>
<tr>
<td>18</td>
<td>Results showing the effectiveness of the filter ensemble and ZeroFedAvg against Data and Model Poisoning Attack</td>
</tr>
<tr>
<td>19</td>
<td>Illustration of a scenario with tampered client modality values</td>
</tr>
</tbody>
</table>
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAGR</td>
<td>Compound Annual Growth Rate</td>
</tr>
<tr>
<td>CAS</td>
<td>Central Aggregation Server</td>
</tr>
<tr>
<td>CCPA</td>
<td>California Consumer Privacy Act</td>
</tr>
<tr>
<td>CDF</td>
<td>cumulative distribution function</td>
</tr>
<tr>
<td>CDP</td>
<td>The Central Differential Privacy</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolution Neural Network</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial-of-Service</td>
</tr>
<tr>
<td>DP</td>
<td>Differential Privacy</td>
</tr>
<tr>
<td>EEA</td>
<td>European Economic Area</td>
</tr>
<tr>
<td>EOCL</td>
<td>Ensemble of Classifier</td>
</tr>
<tr>
<td>EU</td>
<td>European Union</td>
</tr>
<tr>
<td>FE</td>
<td>Filter Ensembles</td>
</tr>
<tr>
<td>FedAvg</td>
<td>Federated Averaging</td>
</tr>
<tr>
<td>FedProx</td>
<td>Federated Proximal</td>
</tr>
<tr>
<td>FedSGD</td>
<td>Federated Stochastic Gradient Descent</td>
</tr>
<tr>
<td>FL</td>
<td>Federated Learning</td>
</tr>
<tr>
<td>FTL</td>
<td>Federated Transfer Learning</td>
</tr>
<tr>
<td>GAN</td>
<td>Generative Adversarial Network</td>
</tr>
<tr>
<td>GDPR</td>
<td>General Data Protection and Regulation</td>
</tr>
<tr>
<td>HE</td>
<td>Homomorphic Encryption</td>
</tr>
<tr>
<td>HFL</td>
<td>Horizontal Federated Learning</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>non-IID</td>
<td>non-independent and identically distributed</td>
</tr>
<tr>
<td>PS</td>
<td>Parameter Server</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>RF</td>
<td>Random Forest</td>
</tr>
<tr>
<td>SGD</td>
<td>Stochastic Gradient Descent</td>
</tr>
<tr>
<td>SMPC</td>
<td>Secure Multi-Party Computation</td>
</tr>
<tr>
<td>TEE</td>
<td>Trusted Execution Environments</td>
</tr>
<tr>
<td>VFL</td>
<td>Vertical Federated Learning</td>
</tr>
<tr>
<td>ZKP</td>
<td>Zero-Knowledge Proof</td>
</tr>
<tr>
<td>ZT</td>
<td>Zero-Trust</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

In recent years, Federated Learning (FL) has received significant attention as a promising approach to machine learning in decentralized environments. The global market of FL is predicted to grow at the Compound Annual Growth Rate (CAGR) of 10.2 percent to 133.1 million by the end of 2032 [1] [2]. This surge of internet-connected devices and strict privacy regulations underscore the need for techniques that allow collaborative model training while preserving data privacy. Traditionally, machine learning requires data to be collected and aggregated on a central server [3]. While this approach works for some data, it becomes problematic when dealing with sensitive personal information. Such data may not be accessible for machine learning purposes because the central server cannot be trusted and may pose a risk of insider threats. Additionally, regulations such as the General Data Protection and Regulation (GDPR) and California Consumer Privacy Act (CCPA) further restrict access to this data, making the data collection process tedious and sometimes almost impossible [4]. In 2016, research scientists at Google first proposed federated learning for machine learning to address these challenges [5]. The core idea of federated learning is to move the computation to the data sources instead of moving the data from the data source to the server. This ensures that the data never leaves its source while still enabling the machine learning model building process. It offers a compelling solution to these challenges by allowing multiple parties to collectively build robust machine learning models without compromising the confidentiality of their data by decentralizing the training process and keeping data local. It minimizes the risk of data breaches and unauthorized access, making it particularly appealing for applications in sensitive domains such as healthcare, finance, and telecommunications. However, despite its potential benefits, FL presents its own set of privacy and security concerns. The following section provides an overview of this research’s motivation, identifies prior research gaps, and discusses the associated research challenges.
Research Motivation

Federated Learning offers a compelling solution for building machine learning models in a distributed, collaborative manner without transferring raw data to a centralized server, thereby addressing data privacy concerns. The following key points highlight the motivations and challenges that need to be addressed:

1. Data Privacy Concerns: Regulations like GDPR [6] in Europe and CCPA [7] in California mandate stringent data privacy measures. It is crucial to ensure that personal or sensitive data is not exposed or misused in federated learning.

2. Data Security Concerns: Federated Learning involves multiple devices and entities, increasing the attack surface. Ensuring a robust security mechanism to prevent data breaches is essential [8]. Additionally, as federated learning exchanges model parameters between clients and the server, secure communication is necessary to prevent interception and tampering.

3. Model Integrity Concerns: The model developed using federated learning should be robust against attacks. Poisoning attacks can alter training data or model updates, leading to biased or inaccurate models. Ensuring the integrity and reliability of the model is paramount.

Research Gaps

Prior studies have shown that data privacy and security in federated learning can be improved by implementing various techniques such as Differential Privacy (DP) [9], Encryption Algorithms (including Homomorphic Encryption (HE) [10] and Secure Multi-Party Computation (SMPC) [11]), hardware-based solutions like Trusted Execution Environments (TEE) [12], and Anomaly Detection Systems (such as FLTrust [13] and FLRAM [14]). However, most of these studies have not combined the aspects of privacy and security, often suffering from the privacy-utility trade-offs. Additionally, Zero-Trust inspired continuous verification of clients on the central server has not been extensively studied in the context of federated learning. This leaves
significant research gaps in developing a comprehensive approach that ensures both privacy and security without compromising the utility of the federated learning models. These research gaps are studied briefly in the related work chapter.

**Research Challenges**

Research to build a secure and privacy-preserving FL framework presents several challenges. These challenges can be summarized in the following points:

1. Development of FL System: This research requires an FL system to run simulations, test, and validate the proposed solution and algorithm. Numerous FL systems are available, such as TensorFlow Federated [15], NVIDIA Flare [16], FLUTE [17], Flower [18], PySyft [19], OpenFL [20], Substra [21], and FATE [22]. Selecting the appropriate framework is challenging, so various frameworks were studied and tested. The Flower framework was chosen for its research-friendly functions and APIs.

2. Implementation of Attack Simulation: One of the most challenging aspects of this research was building the attack simulation framework. Existing frameworks, such as Blades [23] and RobustTailor [24], were studied but found unsuitable as they do not work on top of research frameworks like Flower [18] or FLUTE [17]. These systems are difficult to configure, not easily expandable, and lack diversity in attack types. Therefore, a customizable attack simulation feature was developed on top of the Flower framework to conduct this research.

3. Data Distribution: Distributing data to all clients for FL attack simulation is challenging. Determining how to distribute data among clients effectively is critical. Pathological data distribution was used to create non-iid-ness in the data for this research.

4. Design and Development of Secure FL framework: The federated learning system has many issues related to privacy and security. Developing a novel solution that surpasses existing solutions is a significant research challenge.
5. Investigating the Effectiveness of the System: Evaluating the proposed solution is crucial. Developing effective methods to investigate the FL system’s performance is one of the research challenges.

In light of these challenges, this thesis embarks on a comprehensive analysis of the privacy and security issues inherent in FL. By delving into the intricacies of FL protocols, communication mechanisms, and model aggregation techniques, it aims to identify vulnerabilities and shortcomings that could compromise the privacy and security of FL systems. Furthermore, it proposes a novel ensemble of classifier filter ensemble and Zero-Trust (ZT) inspired continuous verification of client updates in the central server designed to enhance the resilience and privacy-preserving capabilities of FL framework. This ensures that communication is secured to prevent interception and tampering, protects personal and sensitive data from exposure and misuse, and maintains the integrity and reliability of the model. This thesis provides a comprehensive overview of the FL system, beginning with a background study to establish foundational knowledge. It then delves into related works. Significant focus is placed on data privacy and security, emphasizing the importance of adhering to data rules and regulations. Following this, an overview of the proposed solution is presented, along with the experiments and results. Finally, a discussion of the findings, future works, and a conclusion with the implications of this research are provided.
Chapter 2
Background Study

FL is a revolutionary machine learning approach designed to train a global machine learning model across decentralized edge devices that contain local data without sharing those local data with the central server. This approach addresses several critical concerns of the classical centralized machine learning process, including privacy & security of the data being used and bandwidth efficiency. The concept was first introduced in 2016 by McMahan et al. [5] as a collaborative machine learning technique and has since seen widespread interest across various sectors, including healthcare, finance, and telecommunications, due to its ability to leverage data while respecting user privacy and regulatory requirements.

(a) Classical Learning Approach
(b) FL Approach

Figure 2: Comparison between classical and FL Approach

The classical machine learning approach requires the data to be collected on the server prior to the training of the machine learning model, whereas in the FL approach, the data never leaves its source, and only the trained parameters are collected on the server.

Key components of Federated Learning

The FL consists of several key components that work together to enable collaborative Machine Learning (ML) without centralizing the data into the central server. Following are the
primary components of FL.

Central Aggregation Server (CAS)

The CAS, also known as Parameter Server (PS), coordinates the FL process. It is responsible for initializing the global model, selecting the clients for the training, distributing the global model parameters to the clients, collecting the trained model parameters from the clients after the local training, and then aggregating trained parameters to build a new global model. The Central Aggregation Server (CAS) is typically very powerful since it orchestrates clients and aggregates their updates into a single global model. Also, the security measures implemented on the CAS are determined by the governing organization.

Clients (Nodes)

These are the devices where the data originates and is stored. Clients can be smartphones, personal computers, or even servers in different geographical locations. Clients are typically distributed across various locations and are equipped with their own local datasets. The role of each client is to download a copy of the global model and train it using their privately stored local data.

Global Model

This is the ML model developed on the CAS by aggregating all the clients updates after the local training. The global model is updated iteratively through the multiple server rounds, with each server round consisting of sending the model to the clients, performing local training on the clients (local round), and then aggregating the updates centrally.

FL Algorithms

These are the algorithms that make the federated learning process collaborative across multiple decentralized devices without requiring the actual data to be shared with the central
server. This defines the overall learning process. The following are some of the common federated learning algorithms.

Federated Stochastic Gradient Descent

The Federated Stochastic Gradient Descent (FedSGD), also known as sync-SGD [25], is the baseline algorithm of FL. This algorithm was adapted from the traditional Stochastic Gradient Descent (SGD) algorithm to make it work on the FL setup. It requires clients to compute gradients using SGD iteratively using a subset of the local data at each step and send the computed gradients to the CAS. The server averages these gradients to make a global model. This algorithm sends computed gradients after processing each batch of local data on the client, resulting in frequent round trips between the client and the server since the local training round is limited to one. This leads to higher communication costs and can become inefficient when numerous clients are involved in the FL network.

Federated Averaging

The Federated Averaging (FedAvg) [5] is the optimized version of FedSGD algorithm. This reduces the communication overhead by allowing clients to perform multiple local training rounds on their data before sending updates to the server. Clients send model parameters, not just gradients like in FedSGD, after completing a certain number of local training rounds. FedAvg performs well in practice, but it still has issues. It assumes that all participating devices will complete the specified number of epochs of local Stochastic Gradient Descent (SGD), but in FL network there might be stragglers and those can slow down the speed of convergence [26]. To mitigate this, FedAvg often excludes stragglers from the aggregation process. Additionally, FedAvg takes a weighted average of the updates, where the weights depend on the number of data samples from each client device. Hence, it makes the global model more biased towards those devices with more data than the others [26]. There are several optimized algorithms of FedAvg available to make the shared model learn a lot fair. One of them is q-FedAvg [27] which penalizes the worst performing clients more
by some factors. The Per-FedAvg [28] is also the extended variant of FedAvg which seeks to train a model that can be personalized to each device after running a few steps of local gradient descent.

Federated Proximal

The Federated Proximal (FedProx) [29] is an optimization to the FedAvg algorithm, where it adds a regularization term or proximal term to the local loss functions to handle the system heterogeneity better by penalizing large changes in weights by some factor. This helps in dealing with clients that have data that may not be identically distributed or clients that may experience system issues (stragglers) like differing computational speeds or dropping out of the network.

Types of Federated Learning

FL can be classified into three categories based on the distribution characteristics of the data [30]. These are described briefly in the following subsection:

Horizontal Federated Learning

Horizontal Federated Learning (HFL) is a type of FL where the dataset of the clients or the participants will have the same feature space but different samples. For instance, two hospitals in other regions may collect the same types of health metrics from different sets of patients.

Vertical Federated Learning

Vertical Federated Learning (VFL) is a type of FL where the dataset of the clients or the participants will have different feature space for the sample data sample. For example, a bank might have different types of data (features) about the same set of customers.

Federated Transfer Learning

Federated Transfer Learning (FTL) is a type of FL where the dataset of the clients or the participants differ not only in samples but also in feature space [30]. For example, consider a retail
store located in the United States and a hospital located in Nepal. Due to geographical restrictions, there is only a small overlap between the customer bases of these two institutions. Additionally, because they operate in different industries, only a small portion of their feature spaces will overlap.

Moreover, based on network topology, FL can be categorized into two distinct types: centralized and peer-to-peer. Figure 2 (b) depicted above illustrates an example of centralized FL, in which direct communication between clients is not allowed. In this setup, a central server orchestrates the learning process by initiating model training, aggregating the updates from clients, and constructing a global model. Conversely, the peer-to-peer FL operates without needing a centralized server, allowing communication directly between participating clients in the FL network.

Core Principles of Federated Learning

The core principles of FL revolve around a collaborative yet decentralized approach to machine learning, emphasizing privacy, security, and efficiency. These principles enable FL to address some of the most pressing challenges in the field of machine learning, particularly in scenarios where data privacy is paramount, and direct access to data is restricted or impractical.

![Figure 3: Collaborative learning for the next-word predictions on mobile phones using federated learning](https://arxiv.org/abs/1908.07873)

Collaborative Learning

The essence of FL is to enable a form of collaborative intelligence where models are improved collectively by learning from data distributed across numerous devices [31]. This
collaboration occurs without compromising the privacy of the data on each device, as only model updates are aggregated to enhance the global model. Through this collaborative effort, FL can achieve high accuracy and model performance comparable to models trained on centrally housed data, all while upholding stringent privacy standards.

**Privacy and Security**

FL ensures that the data remains on local devices (clients) and is not shared or transferred across the network, and only model updates (such as gradients or parameters) are shared with the CAS for the aggregation, which significantly reduces the risk of exposing sensitive private information. The clients those participated in the learning process do not have any idea about the data that resides on the other client, unlike the classical approach in Figure 2(a) FL keeps the privacy by holding the local data into its own local storage as shown in Figure 2(b). Also, the attack surface is substantially reduced for the data breaches since the data remains on local devices, and only model updates are communicated. FL frameworks often employ sophisticated cryptographic techniques, including SMPC [11] and HE [10], to enhance the security of the model training process and guard against malicious attacks that aim to infer sensitive data from shared updates.

**Decentralized Data**

FL emphasizes the concept of data decentralization [32], the data never leaves from any of the clients and no client has any idea about the data of the other clients. This method not only bolsters privacy and minimizes the risk of data breaches but also honors data ownership and compliance requirements by ensuring data stays within its originating source. The principle of decentralization of FL is especially beneficial for inter-organizational collaborations where there are restrictions on data sharing.
Challenges in Federated Learning

Although FL was developed to create a privacy-preserving machine learning framework, it has several challenges and issues that complicate its application and effectiveness [33]. Some of the major challenges are discussed below:

**Scalability and Communication Overheads**

One of the primary challenges facing FL is scalability. As the number of participating devices in an FL network increases, so does the complexity of coordinating these devices and managing their contributions. Each device in the network contributes updates based on its local data, requiring sophisticated algorithms to aggregate these updates efficiently [34]. Moreover, the communication overhead involved in transmitting model updates from a potentially vast number of devices can strain network resources, despite the updates being significantly smaller than the raw data.

**Data Heterogeneity**

The heterogeneity of data across participating client devices introduces another layer of complexity. Unlike classical decentralized machine learning where the client mostly uses iid data, data from a specific dataset e.g. once client can have class 0 and 1 from MNIST and the other client may accommodate data for the classes 5 and 6 and so on. The FL dataset are typically non-iid in nature [35], the private data of a specific data has no relation with other clients private data. This non-iidness of the private data increases the FL complexity. In real-world applications, the data collected by individual devices can vary widely in terms of quality, quantity, and relevance. This diversity can lead to challenges in ensuring that the aggregated model performs well across all devices, potentially compromising the model’s accuracy and utility. Balancing the model’s performance while accommodating the wide range of data characteristics is a nuanced challenge that FL must navigate.
Data Privacy and Security

Despite the privacy-preserving intent of FL, there are still concerns regarding data leakage through model updates. Inference attacks, such as model inversion attacks, can potentially reveal sensitive information about the training data. Also, malicious actors could potentially target the aggregated model updates or manipulate the learning process through compromised devices. Such attacks could aim to infer sensitive information about data being used or degrade the overall model’s performance.

Convergence Problem

By nature, FL is structured as a bi-level programming problem [36] where the upper level, represented by the CAS, has a single minimization problem aimed at optimizing the global model based on updates received from clients. The lower level consists of multiple minimization problems, each handled by different active clients working with their respective local datasets. Each client’s task is to locally minimize a loss function based on its data before sending the trained updates to the central server. Bi-level programming [37] problems are inherently complex and are classified as NP-hard, indicating that they do not have a straightforward, polynomial-time solution. This complexity is further compounded in large-scale federated learning scenarios, which may involve millions of clients. Each of these clients possesses its unique, non-non-independent and identically distributed (non-IID) data, which adds to the diversity and complexity of the overall optimization problem.

Even though FL comes with several challenges, this thesis will focus on privacy and security issues. FL keeps data private by processing it on local devices instead of sending it over to the centralized server as shown in the Figure 2(b). However, this approach can lead to new types of privacy and security risks, like data poisoning and model poisoning attacks. This thesis will look into these issues, aiming to understand them better and implement new ways to make FL more secure and reliable when it comes to protecting the privacy of the data and keeping the learning process safe.
Chapter 3
Related Work

In the last chapter we have already mention that our motive of the study is to deal with the privacy and security issues of the FL, hence in this chapter we will briefly touch upon the recent works on the security and privacy issues of the FL. There are several research studies performed on addressing privacy and security issues in FL. Most of the researches are grounded in server-side statistical analysis, where the degree of divergence of client models from the global model is compared [13] [38] [39] [40]. Additionally, hardware-based solutions are proposed by employing TEE within processors to secure federated computations [41] [12] [42]. Other approaches, such as local differential privacy (LDP) [43], HE [10], and SMPC [11], in FL also aim to mitigate privacy and security concerns within the framework. However, these solutions often face trade-offs, such as the privacy-utility trade-off [44], where enhancing one aspect often compromises another.

Statistical Anomaly Detection Approaches

Cao et al. [13] proposed a statistical analysis approach called FLTrust, a Byzantine Robust FL Aggregation Algorithm, which enhances the trust in participating clients through a meticulously developed server model based on a clean dataset. This model is compared with the one from the client, and the trust score for each client is calculated and assigned based on the deviation from the client-server divergence threshold. The trust scores are then used to determine whether a client’s parameters will be aggregated into the global model. The divergence between the client model updates and the global model is measured using ReLU-clipped cosine similarity. Their experiments show that they have used 100 clients for six different datasets across six different attack scenarios, and with FLTrust, the attack success rate is less than 10 percent for all different attacks. They achieved promising results in their experiments; however, this approach might not be efficient in scenarios involving a large number of participants with non-IID datasets. The heterogeneity of data can lead to instances of concept drift among clients in the network, where
the patterns and relationships previously learned by local models undergo significant changes, potentially leading to poor performance. In such instances, updates from these clients may not meet the predefined threshold limits set by statistical approaches like FLTrust, resulting in the exclusion of their updates from the global model under the assumption that they are malicious. The Figure 4(a) illustrates the scenario of homogeneous data distribution, which represents the optimal case for heterogeneous data distribution, where each benign client’s update falls within the accepted threshold. On the other hand, Figure 4(b) depicts the worst-case scenario, in which a client may experience concept drift, leading the anomaly detection algorithm to incorrectly tag the client (marked red) as malicious despite being benign. The research by Panchal et. al. [45] shows the proof of the concept drift problem in a FL environment. Also, it is hard to determine the threshold that divides the benign and malicious clients in the network using statistical adversary detection methods.

![Figure 4](https://www.youtube.com/watch?v=KvyzM69PsqM)

**Figure 4:** Concept drift in client due to data heterogeneity across time

*Image Source: https://www.youtube.com/watch?v=KvyzM69PsqM*

FLTrust [13] is related to our work as it also aims to address privacy issues in FL during the training phase and seeks to strengthen the FL environment against different attacks. However, our approach involves implementing solutions on both the client and server sides and does not rely on the statistical divergence between client and server parameters and covers the discussed limitations.

Another defense against Byzantine poisoning attacks called FLRAM [14] proposed by Chen et al. employs isolation forests and improved DBSCAN techniques for anomaly detection.
Isolated forest is an unsupervised learning technique for detecting anomalies. It analyses the degree of isolation of samples in the feature space by implementing binary tree structure of isolate samples. The idea is the anomalous samples will have some more distance from regions of higher data density. The DBSCAN is used to discover clusters of feature spaces to form the data density. This also relies on the statistical difference between the distance of clusters.

There are probability-based aggregation algorithms to secure the FL architecture. Marc Vucovich, Devin Quinn, et al. [46] developed FedBayes: A zero-trust FL aggregation to defend against adversarial attacks, which is based on the probability-based aggregation algorithm that not only enhances the model but also tries to address the potential threat of malicious clients attempting to corrupt the global model. It is intended to be used with a pre-trained model that the server will use to send the initial parameters to the clients as well as compute the mean and standard deviation for the weights of the initial model, denoted as the global mean ($\mu_{\text{global}}$) and global standard deviation ($\sigma_{\text{global}}$). Utilizing these global statistics, FedBayes constructs the global normal distribution and subsequently generates the global cumulative distribution function (CDF). By leveraging this CDF, the algorithm calculates the cumulative probability of receiving a client’s parameters given the prior parameters. This probability computation involves subtracting the client’s weight CDF from the prior weight CDF, with a resultant value closer to zero indicating similarity to the global weights. To mitigate the impact of malicious clients, FedBayes introduces a penalty factor that scales proportionally with the deviation from unity in the probability computation. Experimentation reveals that the differences between malicious and benign clients’ weights are often negligible, prompting the application of a penalty factor of 100 for each percent deviation from unity. Finally, to aggregate the weights, FedBayes adjusts each client’s weights by the computed probability and normalizes the aggregated weights by their sum, effectively enabling the server to remove contributions from malicious clients while preserving the integrity of the global model. LearnDefend proposed by Purohit et al. [47] is also a statistical solution to defend against malicious clients. They use a carefully crafted defense dataset to learn importance score for the client model update. This score will be used to reject the update from
being aggregated into the global model if it is relatively low. The probability-based solution is not entirely reliable as its effectiveness depends on the specifics of the attack scenario. If the attack is engineered to remain below the threshold limit, this defense mechanism can be circumvented. We’re considering these limitations and enhancing the defense mechanism of FL in our algorithm.

**Secure Aggregation Approaches**

Ma et. al [48] also worked on safeguarding privacy and security in the FL framework, where they explored numerous challenges associated with privacy and security in FL and also, presented simulation results to facilitate discussion and propose potential solutions. Their work outlines three broad categories of protective measures against privacy and security issues. The first involves privacy protection on the client side, which includes techniques like adding noise to the generated parameters through differential privacy to obscure sensitive attributes, and sending dummy model parameters alongside the true ones to conceal the client’s contributions during training. The second category focuses on privacy protection on the server side, discussing the development of secure aggregation methods and the implementation of secure multi-party computation, which prevents the server from inspecting individual client updates. The third category pertains to security protection within the FL framework, where they explored homomorphic encryption and back-door defense mechanisms. This also presents experimental results and solutions for four major issues of FL. The client-side approach to preserving privacy through differential privacy may lead to poor model convergence. Therefore, they conducted experiments and suggested investigating the theoretical relationship among noise scale, local training iterations, and the number of communication rounds to address the privacy-utility trade-offs because a higher privacy level will increase more noise on the system. Data poisoning is another issue they discussed, and they proposed three solutions, which were identifying malicious clients through machine learning techniques, developing a secure model aggregation algorithm, and establishing a social relationship between each client and the overall system performance. Another issue addressed was the scaling challenge, where the server may experience long wait
times during parameter uploading. To address this, they proposed setting an upload delay deadline for each client. The server attempts to collect parameters from clients, and if the waiting time exceeds the set deadline, the parameters from that client is abandoned. Lastly, they proposed designing an intelligent aggregator that incorporates a testing process on the server and adjusts aggregation weights based on testing accuracy. These solutions are just propositions to the issues and are not ready for practical application; they do not offer specific solutions to these problems. As presented in their work, we will develop both client and server-side solutions to address the privacy and security issues, creating our own algorithm and process for this purpose.

Li et al. [49] developed a server-side algorithm to remove malicious client updates using spectral anomaly detection, which detects abnormal model updates based on their low-dimensional embeddings. In spectral anomaly detection, high-dimensional data is transformed into lower-dimensional space using techniques such as t-SNE (t-distributed stochastic neighbor embeddings) or autoencoders. The aim of this transformation is to extract only the relevant information for comparison. In their paper, the authors embed both normal and abnormal data instances into a lower-dimensional latent space, where their embeddings exhibit significant differences. This process helps eliminate noisy features from the data instances, allowing abnormal instances to be easily identified by examining reconstruction errors. While this approach does not directly compare statistical divergences between the global model update and client model updates as in FLTrust [13], it does focus on detecting how effectively the client update can reconstruct the benign server update. If the reconstruction is poor, it indicates an anomaly.

Nguyen et al. [50] proposed a secure aggregation process to prevent poisoning attacks on the client as well as the global model parameters protection for the honest-but-curious server. They used HE for the server to combine local model updates and used Zero-Knowledge Proof (ZKP) protocol to detect poisoned clients. These secure algorithms HE increase computational overhead to the FL environment. Also, setting up and maintaining a system that utilizes ZKP increases the complexity of the implementation. Communication overhead is the major challenge if secure aggregation is the choice to protect against poisoning attacks. Zhao et al. [51] proposed a
grouped secure aggregation algorithm to protect against poisoning attacks with low communication overhead. However, their research is limited to preventing data poisoning attacks. Cai et al. [52] proposed an Efficient and secure verifiable federated learning with a privacy-preserving (ESVFL) framework by developing a method to efficiently encrypt the clients’ local gradient since existing secure aggregation solutions were suffering from the computation overhead.

**Hardware Based Approaches**

The PPFL (Privacy-preserving FL) framework developed by Mo et al. [53] also proposed client and server-side solutions to privacy and security issues in FL. The main idea of their approach is to implement a trusted execution environment available on the processor for FL processing. TEEs allow secure storage and execution of arbitrary code on an untrusted device, almost at native speed, through secure memory compartments. Quoc et al [54] proposed a framework called SecFL using TEE which also does the global and local training inside the TEE enclaves to protect the integrity and confidentiality model being developed using the FL. FLATEE is another work proposed by Mondal et al. [41], which also utilizes the TEE environment to isolate the execution of code and handle the data. Zhang et al. [55] also proposed and implemented TEE based federated privacy protection method and extended the support for HFL. This helps secure the model parameters from manipulation by outside sources. However, TEEs have limitations on the Trusted Computing Base (TCB) [56], where not all layers may be loaded into the TCB due to memory constraints. Consequently, techniques such as greedy-layer wise training and aggregation are required, potentially leading to suboptimal training performance and slow convergence. Additionally, implementing FL (FL) into the TEE environment necessitates prior knowledge. Our approach also involves implementing solutions for clients and servers, but with easier implementation and without limitations imposed by the processor’s code base on the learning process.
Filter Ensemble Based Approaches

Various studies have investigated the use of filter ensembles as a strategy to mitigate the impact of adversarial attacks in machine learning. This approach is based on the premise that combining multiple models can enhance system robustness against such vulnerabilities. Xiaoyu Cao, Jinyuan Jia et al. [57] introduced a unique filter ensembling strategy designed to enhance the robustness of machine learning models in a scenario involving multiple data owners, some of which may be malicious. In their proposed method, given $m$ data owners, with $t$ identified as malicious, they construct an ensemble composed of $\binom{m}{k}$ global models. These global models are trained on datasets aggregated from a distinct set of $k$ clients. Despite the innovative approach to mitigate the influence of malicious data owners, this strategy exposes a vulnerability to data reconstruction attacks. This weakness arises because aggregating data from multiple clients to train each model in the ensemble could allow adversaries to reverse-engineer or infer sensitive information from the aggregated datasets, compromising data privacy and security.

Zhang et al. [58] proposed a Kalman filter-based [59] detection solution called FLTracer to identify malicious clients by observing the behavior changes before and after the attack. They also provided strategies to trace the attack time, objective, type, and poisoned location of the model update. Dasgupta et al. [60] introduced dual-filtering (DF) schemes for learning systems to defend against adversarial attacks. Their approach incorporates filters at both the input stage (before data samples are fed into the core learning model) and the output stage (before the decision component). The input filter eliminates misleading and out-of-distribution data, while the output filter addresses large variations and restricts misclassifications to enhance the model’s overall accuracy. This is the standalone ML solution, however this could be a promising solution for FL to protect against such adversarial attacks.

In this thesis, we will employ a filtering scheme similar to the input filtering approach proposed by Dasgupta et al. [60] and utilize that as part of a filter ensemble to protect against data poisoning attacks for the FL environment.
Chapter 4
Data Privacy and Security

Data privacy and security involve protecting information from unauthorized access and handling it responsibly. Privacy focuses on collecting, using, and sharing personal data, aligning with individuals expectations and legal requirements, like the GDPR [6], CCPA [7] etc. Security addresses protecting data from threats like cyberattacks through measures such as encryption and secure authentication. Both are crucial for maintaining trust and safeguarding data confidentiality and integrity in the digital age. FL aligns with privacy-preserving objectives by limiting data exposure and access. Also it tries to protect the model and its updates from unauthorized access, manipulation, or theft as these updates are communicated across the network using techniques such as encryption, secure aggregation etc.

General Data Protection and Regulation (GDPR)

The GDPR is a regulation in European Union (EU) law on data protection and privacy in the EU and the European Economic Area (EEA). It also addresses the transfer of personal data outside the EU and EEA areas [61]. Any organization that handles personal data must comply with the seven key principles of the GDPR. These are: lawfulness, fairness, and transparency; purpose limitation; data minimization; accuracy; storage limitation; integrity and confidentiality; and accountability [62]. These compliances adds complexity to the development of ML models whenever the nature of the data to be used is private and sensitive. This is where FL simplifies the compliance, as the original data never leaves its source in FL, making it easier to adhere to the GDPR requirements.

California Consumer Privacy Act (CCPA)

The CCPA [7] is the privacy act to enhance privacy rights and consumer protection for residents of California, United States. This is the consumer act, and the consumer has the right
to know about the data being collected about them, the right to access their own data, the right to opt-out and opt-in of the sale of their personal data, the right to request the deletion of their data [63]. The FL helps to facilitate compliance with CCPA by minimizing the data exposure since the original data never leaves its source, providing data control and transparency since the consumer will have full control over their data and so on.

**Existing Privacy and Security Attacks**

As observed in the previous chapters (2 and 3), data privacy and model integrity are paramount in the FL system. Therefore, understanding the nuances of various privacy and security attack methods is crucial for developing robust defense system. Below are some of the existing attacks:

*Inference Attacks*

An inference attack [64] is a type of attack in FL where the adversary tries to extract the sensitive information from the model without directly accessing the actual data. In FL approach, multiple entities collaboratively train a model while keeping their data local, aiming to preserve privacy. However, attackers can still infer private information by analyzing the statistical information in model updates shared across the network. Inference attacks can be classified into three main types: Model Inversion Attacks, Property Inference Attacks, and Membership Inference Attacks, each of which is described below:

**Model Inversion Attack**

This is a reconstruction attack where attackers reconstruct input data from model outputs or parameters [65] [64]. This poses significant privacy risks in applications where training data may contains personal or confidential information. This attack exploits the model’s ability to learn and generalize from its training data by carefully analyzing the model’s responses to various inputs. The attackers can deduce patterns, features, or data points that were likely present in the training
set using this attack. In essence, the attacker reverse-engineers the model’s learning process to reveal sensitive information about the data on which it was trained.

Property Inference Attack

This is an inference attack where an attacker aims to uncover global properties of the dataset that were not explicitly shared [66]. Unlike direct attacks that attempt to steal or replicate individual data points, property inference attacks focus on learning broader characteristics or patterns about the aggregated data from the FL model. For example, while building a machine learning model using healthcare data, an attacker in the network might analyze the aggregated model updates to infer sensitive demographic properties about the patients in the training set, such as the distribution of age groups or diseases in the training set.

Membership Inference Attack

This is a type of inference attack in which the attacker identifies whether specific data points were used in the training phase, posing a significant privacy risk [66]. Despite FL’s intent to protect individual data by sharing only model updates among participants, member inference attacks exploit the model updates to infer data participation. Unlike other attacks that also utilize model outputs, the attacker’s intent is to verify if the data they possess was included in the victim model’s training dataset, highlighting a nuanced yet potent threat to data privacy in FL environments.

Figure 5: Example of membership inference attack
Poisoning Attacks

Poisoning attack [67] is the type of attack in which malicious participants alter their local datasets to include incorrect or misleading information. When these corrupted datasets are used for local model training, the resulting model updates can skew the global model in harmful ways, affecting its accuracy or introducing targeted biases. In general, poisoning attacks can be categorized into two classes: untargeted attacks and targeted attacks. Untargeted attacks [68] are designed to disrupt a system’s normal functioning, aiming to reduce its overall effectiveness and undermine reliability without targeting a specific outcome. In contrast, targeted attacks [68] seek a precise, predetermined result, requiring a comprehensive understanding of the system’s vulnerabilities to manipulate it into yielding a specific incorrect output. Poisoning attacks can be further categorized into Denial-of-Service (DoS) and distortion attacks. DoS attacks aim to prolong the training duration, effectively hindering the learning process, while distortion attacks corrupt the model, causing it to deviate significantly from its intended behavior.

These attacks, encompassing targeted, untargeted, DoS, and distortion types, can be broadly categorized into model poisoning and data poisoning attacks. Both categories can manifest as targeted, untargeted, DoS, or distortion attacks, affecting either the integrity of the data used for training or the structure and functionality of the model itself. In this thesis, we will primarily focus on the model as well as the data poisoning attacks and develop a novel algorithm to fortify against these attacks on the FL framework.

In Figure 6, if the server becomes unaware of poisoned data or a poisoned model, the regular FL process will still aggregate the poisoned parameters into the global model. Consequently, the global model may then operate according to the intentions of the malicious actor in the network who introduced the poison to the data or model. This underscores the necessity of reinforcing the FL environment against both data and model poisoning attacks to ensure the development of a secure and privacy-preserving FL framework.
Targeted Poisoning Attacks

In targeted poisoning attacks, the adversary’s objective is to manipulate the behavior of the target model so that it produces incorrect predictions for specific inputs. This form of attack involves subtly altering the training data or injecting malicious data into the dataset, with the alterations crafted to achieve a particular malicious outcome without necessarily disrupting the model’s performance on other inputs. For instance, an attacker might target a model trained to recognize handwritten digits with the goal of causing it to misclassify all instances of the digit 3 as 8. To accomplish this, the attacker could introduce poisoned examples into the training dataset—images of 3 that are subtly modified to resemble 8 closely enough to confuse the model, or vice versa, depending on the attack strategy.

Untargeted Poisoning Attacks

In untargeted poisoning attacks, the adversary’s objective is to degrade the overall performance of the target model, also reducing the accuracy of some samples while keeping the other parts as original. This attack typically involves one or more malicious participants which introduce harmful model updates during the training process. These updates are designed not to
steer the model towards specific incorrect predictions but to generally reduce the model’s accuracy and reliability across a wide range of inputs. This could be achieved through various means, such as introducing noise into the model updates or sharing deliberately degraded updates derived from manipulated local training processes.

Figure 7: Venn diagram showing the relationship among the poisoning attacks

Backdoor Poisoning Attacks

In backdoor poisoning attack [69], the adversary implants a backdoor into the target model while keeping the overall performance of target model intact. This can be targeted or untargeted. Backdoor poisoning attack involves one or more of the participating entities in the training process maliciously modifying their local model updates to include the backdoor. This could be achieved by injecting data with a specific pattern or trigger into their training set and associating it with a wrongful output. When these poisoned updates are aggregated into the global model, the model learns to associate the trigger with the attacker’s intended output, activating the backdoor whenever the trigger is present.
Model or Data Poisoning Attacks

Data poisoning attack [70] refers to tampering with the original data, whereas model poisoning attack [70] refers to manipulating model updates or replacing the model before sending them to the central server for aggregation. Model poisoning can be more subtle and difficult to detect than data poisoning, as the model updates may still appear valid but are designed to gradually or suddenly corrupt the global model. Model or Data poisoning attacks can be either targeted or untargeted. One example of a data poisoning attack is a label flipping attack, where the labels are flipped in the training dataset such that the performance of the machine learning model degrades.

Generative Adversarial Network (GAN) Attacks

A Generative Adversarial Network (GAN) attack [71] is an attack where an adversary employs GANs to generate synthetic data or model updates that can undermine the integrity or confidentiality of the FL process. GANs are a class of artificial intelligence models comprised of two networks: a generator that creates data samples and a discriminator that evaluates them for authenticity. GAN based attacks are especially the data poisoning attacks. In the context of FL, GAN attacks leverage this architecture to produce inputs or model updates that are indistinguishable from genuine ones, thereby manipulating the learning process or extracting sensitive information without detection [72]. As we can see in Figure 8, we can easily create fake samples of faces using a generative adversarial network and application. These samples help the adversaries easily create a dataset with the poisoned target label.

Figure 8: Fake images generated by GAN using the website thispersondoesnotexist.com

These potential attacks involve leveraging synthetically generated data that would not
typically be encountered during model training, making GAN-based attacks highly effective in security and privacy contexts.

**Existing Solutions for Privacy and Security Issues**

Numerous studies have been conducted to protect FL against privacy and security issues. One approach is the application of DP, which adds noise to data or updates to ensure individual contributions remain private [73]. The another approach is SMPC [74], and HE [75] allows for secure computations and aggregation of model updates without exposing raw data. Anomaly detection mechanisms identify and mitigate malicious updates, preventing model poisoning. Additionally, integrating blockchain technology enhances transparency and security by providing a tamper-evident record of updates. These strategies collectively improve the privacy and security of FL systems.

**Differential Privacy (DP)**

The DP enhances data privacy in FL by introducing noise to the data or the model updates during the training phase [73]. In FL, various clients work together to develop a global model, and they process updates on their datasets locally before dispatching these updates to a central server for aggregation. DP is implemented by adding a predetermined level of randomness into these updates prior to their aggregation. Such a strategy guarantees that the combined model remains incapable of revealing sensitive details about any participant’s data, thus safeguarding the privacy of individual datasets. For instance, in a FL setup where hospitals aim to collaboratively create a model for predicting patient outcomes from diverse health metrics without directly exchanging patient data, differential privacy is employed. Each hospital integrates noise into its model updates before forwarding them to the aggregator, ensuring the privacy of patient data while still contributing to the development of a predictive model. H. Brendan, Daniel Ramage et al. [9] introduced an adaptation of the federated averaging process that incorporates noise to comply with user-specific DP and explored the application of differentially private SGD for protecting privacy.
on an individual data level. However, this technique overlooks the potential risk posed by central servers that might be untrustworthy or compromised, which could result in privacy violations.

\[ W_{t+1} = W_t + DP(Aggr(\Delta W^t_1 + \Delta W^t_2 + \ldots + \Delta W^t_n)) \]

\[ W_{t+1} = W_t + Aggr(DP(\Delta W^t_1) + DP(\Delta W^t_2) + \ldots + DP(\Delta W^t_n)) \]

(a) Centralized Differential Privacy (CDP) in FL

(b) Local Differential Privacy (LDP) in FL

Figure 9: Illustration of CDP and LDP in FL

The Central Differential Privacy (CDP), as shown in Figure 9(a), assume a large participant base to balance privacy and accuracy appropriately. In such situations with limited participants, FL models utilizing CDP may face challenges in achieving convergence or attaining a satisfactory accuracy level. In the figure 9(a), DP is the differential privacy function, Aggr is the aggregator function and \( \Delta W^t_1 \) is the initial model update, \( W_{t+1} \) is the global updated aggregated using all \( \Delta W^t_n \) local updates. The problem with the CDP is that if the server is honest-but-curious then it can read all the updates coming from the client devices.

The updated version of DP is local differential privacy, as shown in Figure 9(b), where the differential function will be applied to each and every update that comes through the client to the server prior to aggregation. This makes it more secure and lowers the risk of insider threats from the centralized server. DP techniques are combined with other methods to enhance privacy and security in FL.
Secure Multi-party Computation (SMPC)

The SMPC in FL is a cryptographic technique that allows multiple parties to collaboratively compute a function over their inputs while keeping those inputs private [76]. SMPC facilitates the process by ensuring that model updates contributed by each participant are aggregated so that the final computation can be performed without exposing any participant’s raw data. Secure multi-party computation relies on methods such as oblivious transfer, garbled circuits, and secret sharing. The Oblivious Transfer [77] is a protocol where a sender transmits one of many pieces of information to a receiver without knowing which piece was transferred. The Garbled Circuits [76] enable secure two-party computation by obscuring input operations into an indecipherable format except for the outcome. Secret Sharing [76] is a method where a secret is divided among multiple parties, none of whom can access the information alone but can jointly compute functions on the secret without revealing it to the others.

Homomorphic Encryption (HE)

The HE is also the cryptographic algorithm which helps to enhance the privacy and security issue in FL [75]. It is a technique that allows computations to be performed on encrypted local model parameter data, enabling the model to learn from the data without ever decrypting it. This preserves the privacy of the data while still allowing for the collaborative updating of a shared model across different parties or devices.

Trusted Execution Environment (TEE)

TEE is a secure area within a computer’s processor that is designed to safeguard sensitive information and ensure that operations on this information are carried out in a secure manner [78]. TEEs work by isolating the execution of selected code and the processing of sensitive data from the rest of the device’s operating system and applications. This isolation helps protect against software attacks and unauthorized access, making it an essential feature for enhancing the security of electronic devices. This can be used utilized to create a shield for the FL environment.
Various solutions have been proposed to defend against adversarial attacks in the FL environment, including the techniques discussed above. However, the key principle of zero trust security model [79] — ”Never Trust, Always Verify” — in the context of FL for client verification has yet to be explored. Furthermore, much of the research focuses solely on either data poisoning or model poisoning, not both. This thesis proposes methods to defend against both model and data poisoning attacks using continuous client verification and filter ensembles.
Chapter 5
Fortifying FL with Filter Ensembles and Continuous Verification

Data poisoning and model poisoning attacks are common in a FL environment. To fortify FL against these attacks, a comprehensive solution motivated by the ZT principle of ”Never Trust, Always Verify” [79] is implemented. This approach involves continuously verifying clients to detect and mitigate model poisoning. Additionally, input filter ensemble on the client side is implemented to defend against data poisoning attacks.

Continuous Verification (Never Trust Always Verify)

The idea behind the continuous verification of client update is inspired from one of the key principle of ZT. It is a security model that operates on the principle that no entity, inside or outside the network, is trusted by default. This model assumes that threats can originate from anywhere, and thus, every access request, regardless of its origin, must be fully authenticated, authorized, and encrypted before granting access. This approach focuses on preventing unauthorized data and service access by enforcing highly specific access controls. Although the ZT security model covers a wide range of topics, our proposed solution to defend against poisoning attacks is grounded on the only the one core principle of ZT i.e. continuous verification (never trust, always verify).

In the context of FL, continuous verification of clients can prevent model poisoning attacks and bolster the security of the overall FL ecosystem. However, implementing continuous verification within a distributed federated system presents challenges, as any device may participate for learning purposes. This thesis proposes a new method for the continuous verification of clients in a FL environment to protect against model poisoning attacks. The FL system poses a significant challenge in developing a continuous verification mechanism within its framework. The continuous verification on the server requires certain metrics of clients for validation, but in a FL context, the server often lacks information about the clients. To facilitate this process, we have employed the concept of client modalities.
Client Modalities

Client modalities are the various attributes or characteristics that define the clients in the FL network. These modalities should not consist the privacy sensitive information. These modalities can encompass a wide range of features, including but not limited to: device type, operating system, mac address etc. As the purpose of this research, 38 different client modalities are collected from the mobile device. The client modality collection process involves gathering client attributes from the devices. This includes generating sample values and definitions for each client modality, accompanied by a feasibility study to assess the modality’s usefulness.

The appropriateness of the modality, based on privacy sensitivity, is manually assigned a rating from 1 to 10, where 1 indicates not appropriate and 10 indicates highly appropriate in the range. Some of the modalities are shown in the Table 1. The values of all client modalities are collected from the client devices during the initialization phase of the FL environment and stored in the server’s knowledge base, which can be a physical database like MySQL, PostgreSQL, or others.

Symmetric Key Encryption

Symmetric key encryption is a type of encryption where the same key is used for both encrypting and decrypting data [80]. After setting up the FL environment and collecting all client modalities, the server initiates the FL process by generating a modality request ID and other parameters. It then sends that information to the client. The client then performs local training and encrypts the model’s parameters using the key associated with the requested modality value. These encrypted parameters are transmitted back to the server, which decrypts them using the same key, leveraging the modality values stored in its knowledge base.

In the Figure 12, the Gen() function generates a modality id, Mid, and the Net() function computes the initial model parameters and sends those to the client. The client then performs local training to produce W, utilizing their own Net() function on their data. Subsequently, it retrieves the requested
<table>
<thead>
<tr>
<th>Name</th>
<th>Sample Value</th>
<th>Definition</th>
<th>How it helps?</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFi (MAC)</td>
<td>00:1A:2B:3C:4D:5E</td>
<td>It is a globally unique identifier assigned to a wifi adapter.</td>
<td>It is unique to each device</td>
<td>10</td>
</tr>
<tr>
<td>Bluetooth (MAC)</td>
<td>A1:B2:C3:D4:E5:F6</td>
<td>It is a globally unique identifier assigned to a bluetooth adapter.</td>
<td>It is unique to each device</td>
<td>10</td>
</tr>
<tr>
<td>Device ID</td>
<td>a9a4c8c3b51d8463</td>
<td>The device ID is a number generated when the device is first set up. It serves as a unique identifier for that specific device. This concept is similar across iPhones and devices from other manufacturers. It is not always guaranteed to be unique across all devices, and it may change in some cases i.e factory reset</td>
<td>It is unique to each device</td>
<td>10</td>
</tr>
<tr>
<td>Operating System</td>
<td>Android</td>
<td>Operating System</td>
<td>It is common but can be helpful.</td>
<td>8</td>
</tr>
<tr>
<td>Device Serial Number</td>
<td>ABC12345XYZ</td>
<td>A device’s serial number is a unique identifier assigned to each individual device. It is used for various purposes, including warranty tracking and device management.</td>
<td>It is unique to each device</td>
<td>10</td>
</tr>
<tr>
<td>Device Type</td>
<td>Apple iPhone, Google Pixel, OnePlus</td>
<td>Device types for mobiles typically refer to the broad categories or brands of mobile devices.</td>
<td>It can be used along with other attributes.</td>
<td>10</td>
</tr>
<tr>
<td>Device name</td>
<td>iPhone 14 Pro, Google Pixel 6</td>
<td>This is the name of the device.</td>
<td>It can be used along with other attributes.</td>
<td>10</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>Apple, Samsung, Google</td>
<td>This is the name of the manufacturer of the device.</td>
<td>It can be used along with other attributes.</td>
<td>10</td>
</tr>
<tr>
<td>Chipset</td>
<td>Apple A15 Bionic chip with 6-core CPU, Qualcomm Snapdragon 765G</td>
<td>This is the CPU chip that is used in the device.</td>
<td>It can be used along with other attributes.</td>
<td>10</td>
</tr>
<tr>
<td>Memory Capacity</td>
<td>RAM 16GB, RAM 8GB</td>
<td>The is the memory capacity of the device.</td>
<td>It can be used along with other attributes.</td>
<td>10</td>
</tr>
<tr>
<td>Storage Capacity</td>
<td>256 GB, 1TB</td>
<td>This is the storage capacity of the device.</td>
<td>It can be used along with other attributes.</td>
<td>10</td>
</tr>
<tr>
<td>Screen Resolution</td>
<td>1290 x 2796 pixels, 1600 x 2560 pixels</td>
<td>Screen resolution refers to the number of pixels and density, while screen size refers to the screen’s physical dimensions.</td>
<td>It can be used along with other attributes.</td>
<td>10</td>
</tr>
<tr>
<td>Pixel Density</td>
<td>274 ppi, 460</td>
<td>Refers to the concentration of pixels on a particular display, measured in pixels per inch (ppi)</td>
<td>It can be used along with other attributes.</td>
<td>10</td>
</tr>
<tr>
<td>GPU Type</td>
<td>Apple GPU, Adreno 740</td>
<td>This is the graphics processing unit used in the device.</td>
<td>It can be used along with other attributes.</td>
<td>10</td>
</tr>
<tr>
<td>CPU Type</td>
<td>Octa-core, Hexa-core</td>
<td>This is the CPU type used in the device.</td>
<td>It can be used along with other attributes.</td>
<td>10</td>
</tr>
<tr>
<td>Language Settings</td>
<td>English (United States)</td>
<td>This is the default language used in the device.</td>
<td>It can be used along with other attributes.</td>
<td>8</td>
</tr>
<tr>
<td>Time zone</td>
<td>Central</td>
<td>A time zone is an area which observes a uniform standard time for legal, commercial and social purposes.</td>
<td>It will be same for some region/area. If the person with the mobile travels ,then this might get change.</td>
<td>8</td>
</tr>
<tr>
<td>Network card serial number</td>
<td>ABCD-123456-7890</td>
<td>It is a serial number is a unique number assigned to a product by the manufacturer.</td>
<td>It is unique to a network card.</td>
<td>10</td>
</tr>
<tr>
<td>Camera Pixel</td>
<td>48MP</td>
<td>It is the resolution of the camera’s image sensor.</td>
<td>It can be used along with other attributes.</td>
<td>8</td>
</tr>
</tbody>
</table>
modality value, Mv, from the device using the fetch() function. The trained parameters W are encrypted with the key Mv through the Enc() function, resulting in We. This encrypted data W_e is transmitted back to the server, which retrieves the original modality value Mv from its knowledge base with the getMv() function and decrypts the model parameters using Mv as the key, yielding W_n. The original encryption key is never shared across the network.

This is how the federated learning environment protects from both data and model poisoning attacks using our proposed solutions, ensuring the integrity and performance of the model despite the presence of malicious clients.

**Defense Against Model Poisoning**

The client update is continuously verified in the server using the client modalities and symmetric key encryption technique. The client modality value will act as the symmetric key for both the server and client on each global round. If the server cannot decrypt the model parameter using the requested modality id value, then that client update will be flagged as poisoned or tampered update. This protects the FL process against model poisoning attacks.

Suppose in our FL environment of 1000 clients, we have one specific malicious actor: Client B, whose model is poisoned. An adversary attempted model poisoning on Client B by replacing the model parameters and these poisoned parameters are sent to the server. The server couldn’t
decrypt the poisoned model parameters because the adversary did not possess the correct modality value. As a result, the server rejected these updates, preventing them from being aggregated into the global model. This is how our proposed solution defended against model poisoning attacks.

**Defense Against Data Poisoning**

This thesis introduces the concept of filter ensembles as a defense mechanism against data poisoning in FL. In machine learning, filter ensemble is a technique that combines the outputs of multiple filter methods for feature selection to enhance the selection process. Multiple filters are developed and integrated to create a filter ensemble. This technique collects the outputs from all the filters and applies majority voting to determine the final output. It was inspired by the work of
Dasgupta et al. [60] on dual-filtering schemes for learning systems to prevent adversarial attacks. The thesis adopts the concept of an input filter to establish a defense against data poisoning attacks.

Filter ensembles are prepossessed and installed during the FL orchestration process for each and every participating client. The training data first passes through the filter ensemble, which determines the output for the target value based on the majority voting from each filter model. The filtered data is then used for the local learning process. This is how Filter Ensembles (FE) safeguards against data poisoning attacks effectively.

Suppose we have one specific malicious actor: Client A, whose data is poisoned in our FL system. Client A submits data that has been intentionally manipulated to introduce biases and inaccuracies into the training process. To counter this, a filter ensemble was employed in that client device that was pre-trained on the local data itself before any local training on that device. This mechanism processes the input data (poised) and produces the filtered data (corrected), removing any inconsistencies in the dataset. As a result, the corrupted data from Client A is removed, preventing it from affecting the model’s training. This illustrates how our proposed solution
protects FL system against data poisoning attacks.

**Model Evaluation**

The model is evaluated using the accuracy metric, which measures the proportion of correctly predicted instances out of the total instances in the dataset. Accuracy is computed for the filter ensemble, local model, and global model.

\[
\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{1}
\]

This metric provides a measure of the model’s ability to make correct predictions across distributed data sources, reflecting its generalization capability and overall effectiveness.
Chapter 6

Experiment & Results

The primary goal of our experiment was to implement, test, and evaluate the efficacy of integrating a filter ensemble alongside the zero trust concept of continuous verification of clients within a FL framework. This effort was focused on testing how effective these methods are in protecting against data and model poisoning attacks. At the beginning, the CAS sends an invitation to all clients, and interested clients join for collaborative learning. The CAS then sends the initial model parameters and filter ensemble to the clients. All clients train the FE on their private data, and once trained, the private data is processed through this FE for local training. Each client performs the predefined local iterations of training on the filtered data and then it sends the parameter updates by encrypting them using the requested modality value. This is discussed in more detail in the following section.

Framework Setup

The experiment was conducted on the Flower framework [18]. The Flower framework is an open-source framework designed to simplify the implementation and simulation of the FL process. This framework is research-friendly and allows customization of server and client configuration. It consists of a state-of-the-art averaging algorithm built into the system. However, it does not include an attack simulation framework. Therefore, for the experiment, an attack simulation for both model and data poisoning attack was developed.

Simulation Process

The simulation process for this experiment can be summarized as a multi-step process. As illustrated in Figure 13, the process encompasses ten steps described below:

1. Read configuration - The flower framework reads configuration form the base.yaml file.
All the server and client configuration along with client modalities for the simulation are available in the base.yaml file.

2. Prepare dataset - The flower frameworks prepares dataset for all clients based on the configuration given. The original dataset is partitioned to all the clients and to give the non-iidness on the dataset, pathological partitioning is done where every client will get at most certain number of classes in a partition. Also, poisoned dataset is created for configured clients using label-flipping technique. Each client will possess both training and validation datasets. For the purposes of simulating attacks, the training dataset of some clients may be poisoned.

3. Client Provisioning - This process involves setting up the filter ensemble, client modalities, and implementing a symmetric key encryption algorithm in each clients.

4. Setup Averaging Strategy - Two averaging strategies were developed: one is the basic
FedAvg strategy, and the other is FedAvg with continuous verification, which we call ZeroFedAvg. The ZeroFedAvg incorporates the logic of continuous verification.

5. Server Provisioning - The server implements the averaging strategy and makes everything ready for the FL process.

6. Start Simulation - The flower framework starts the simulation.

7. Training - After starting the simulation, the server creates a request modality id and initial model parameters and sends to the client.

8. Data Filtering - On receiving the initial parameters and configurations, the client implements the data filtering using the filter ensemble and filtered data is used as the training dataset. The trained model parameters are encrypted using the request modality value and returned back to the server. The implementation of filter ensemble can be toggled on or off for the simulation purpose.

9. Continuous Verification - The client model parameters are continuously verified on the server using the requested modality value. The server decrypts the model parameters using the requested client modality value. If decryption is successful, it proceeds with aggregation using FedAvg; otherwise, it rejects the update. The implementation of continuous verification can be toggled on or off for the simulation purpose.

10. Stop - Simulation process stops

Dataset

A Modified National Institute of Standards and Technology (MNIST) [81] handwritten digits dataset was used for the experiment. The dataset comprises 70,000 images of handwritten digits (0 through 9), split into a training set of 60,000 images and a test set of 10,000 images. Each image is a 28x28 pixel grayscale image. The training set was further divided to create a validation set for local training in a 90:10 ratio. The training dataset is partitioned for each client using
a pathological partitioning technique such that each partition contains at least five classes. This pathological partitioning technique introduces non-IIDness in the clients’ training datasets.

**Model Development**

The client used the Convolution Neural Network (CNN) [82] for model development. It processed grayscale images (1 channel) and classified them into a specified number of classes (digits 0-9). It was a simple neural network consisting of two convolution layers, one max pool layer, and three fully connected layers. The Rectified Linear Unit (ReLU) [83] activation function was used after each convolution and the first two fully connected layers. The objective of the experiment was to create a secure FL environment, rather than to optimize the neural network architecture. Therefore, a simple convolution neural network was employed for digit classification.

![Neural Network Diagram](image)

Figure 15: Illustration of the Neural Network used for MNIST handwritten digits recognition

Figure 15 is the network architecture used for the MNIST handwritten digits recognition in our experiment. The comprises a series of convolution layers (Conv1 and Conv2), pooling layer (MaxPool1), and fully connected layers (FC1, FC2 and FC2).
Filter Ensemble Development

Different Random Forest models [84] were trained after the initialization of the client to create a FE on the client device using their own dataset. Each filter was tested accordingly; Table 2 shows the validation accuracy of five Random Forest (RF) models trained to create a combined FE on one client device.

Table 2: Random forest model used in the filter ensemble for the experiment

<table>
<thead>
<tr>
<th></th>
<th>Model1</th>
<th>Model2</th>
<th>Model3</th>
<th>Model4</th>
<th>Model5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.993</td>
<td>0.986</td>
<td>0.991</td>
<td>0.992</td>
<td>0.987</td>
</tr>
</tbody>
</table>

The above is one of the FE trained and developed in the client device which shows the overall FE accuracy of 0.965 on their test data.

Experiment Results

The experiment was conducted under different scenarios: one with data poisoning and another with model poisoning. Results were collected both with and without the proposed methods. For data poisoning, this included comparisons with and without the filter ensemble, and for model poisoning, comparisons were made with and without continuous verification of updates using ZeroFedAvg.

Results against Data Poisoning Attack

The experiment was conducted with a global configuration of 300 clients (300 different Flower client instances) in the network, spanning 100 server rounds and 20 local rounds. Model M1 represents the model developed without using the filter ensemble, whereas model M2 represents the model developed using filter ensemble.

Data was poisoned using the label replacement technique, in which every label of the MNIST images in the training set was replaced with 9 in each server round. As we can see in the table
Table 3: Experiment details against data poisoning attack

<table>
<thead>
<tr>
<th>Poisoned Clients</th>
<th>Accuracy (M1)</th>
<th>Accuracy (M2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>2%</td>
<td>96.81</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>4%</td>
<td>95.60</td>
</tr>
<tr>
<td>Experiment 3</td>
<td>8%</td>
<td>89.58</td>
</tr>
</tbody>
</table>

Figure 16: Results showing the effectiveness of the filter ensemble against Data Poisoning Attack

3, we observe the comparative performance of two models, M1 (without filter ensemble) and M2 (with filter ensemble), under increasing levels of client data poisoning within a FL environment. Experiment 1 showed that with 2% of the clients providing poisoned data, model M1 achieved an accuracy of 96.81%, while model M2 performed slightly better with an accuracy of 97.83%. Mostly these model parameters are set of numbers between 0 and 1 and if we add 0.01 and 0.01 to 50 times, there won’t be much significant difference. This is why there is a slight change due to the poisoned client. If we increase the number of clients to a million, then there might be a difference. The proportion of poisoned clients rose to 4% in Experiment 2; it is observed that there was a marginal drop in accuracy for model M1 to 95.60%, whereas model M2 exhibited a small increase in accuracy to 97.89%. The trend became more pronounced in Experiment 3, where with 8% poisoned clients, model M1’s accuracy significantly decreased to 89.58%, while model M2’s accuracy improved to 98.12%. The model M2 performed well for all scenarios of data poisoning, particularly with higher percentages of compromised clients in the network. In the Experiment
3 (without the filter model), model hiccups were observed due to model confusion, which is the model’s inability to correctly distinguish between different classes. This issue was introduced by class overlap, where different classes in the dataset have overlapping features. However, the model protected using the filter ensemble did not suffer from the model confusion.

Results against Model Poisoning Attack

This experiment was also conducted on the global configuration of 300 clients in the network, spanning 100 server rounds and 20 local rounds.

Table 4: Experiment details against model poisoning attack

<table>
<thead>
<tr>
<th>Poisoned Clients</th>
<th>Accuracy (M3)</th>
<th>Accuracy (M4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 4</td>
<td>2%</td>
<td>97.27</td>
</tr>
<tr>
<td>Experiment 5</td>
<td>4%</td>
<td>95.39</td>
</tr>
<tr>
<td>Experiment 6</td>
<td>8%</td>
<td>74.38</td>
</tr>
</tbody>
</table>

Figure 17: Results showing the effectiveness of the ZeroFedAvg against Model Poisoning Attack

The table 4 showcases the performance of two models under model poisoning attacks where the attackers replaces the model parameters with the poisoned model. Model M3 uses the standard FedAvg algorithm, and model M4 employs ZeroFedAvg, which is our proposed solution which does the continuous verification of client updates to defend against such attacks. In Experiment 4, where 2% of the client models were poisoned, Model M3 managed to achieve 97.27% accuracy. Model M4, utilizing ZeroFedAvg, demonstrated greater resilience to the attack, maintaining a
higher accuracy of 98.16%. The resilience of M4 became even more evident in Experiment 5, with the poisoning level increased to 4%. Model M3 experienced a drop in accuracy to 95.39%, while M4’s accuracy only slightly decreased to 97.94%. The trend continued in Experiment 6, with 8% of the models being compromised. M3’s accuracy suffered a substantial decline to 74.38%, whereas M4’s accuracy remained robust, only slightly decreasing to 98.02%. Similar Model hiccups were seen while introducing an increased number of malicious clients on the network. The consistent performance of Model M4, even as the extent of the attack intensified, underscores the effectiveness of the ZeroFedAvg algorithm in preserving the integrity of the model amidst model poisoning attempts.

**Results against Data and Model Poisoning Attack**

This experiment was conducted to assess the effectiveness of our filter ensemble and ZeroFedAvg performance in response to both data and model poisoning within a FL environment. For this experiment, 300 clients were used with the process spanning 100 server rounds and 20 local training rounds.

<table>
<thead>
<tr>
<th>Poisoned Clients</th>
<th>Poisoned Clients</th>
<th>Accuracy (M5)</th>
<th>Accuracy (M6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Data)</td>
<td>(Model)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experiment 7</td>
<td>2%</td>
<td>2%</td>
<td>96.09</td>
</tr>
<tr>
<td>Experiment 8</td>
<td>4%</td>
<td>4%</td>
<td>50.32</td>
</tr>
<tr>
<td>Experiment 9</td>
<td>8%</td>
<td>8%</td>
<td>56.86</td>
</tr>
</tbody>
</table>

As we can see on the table 5 Two models were evaluated under increasing levels of data and model poisoning, Model M5 without any defense mechanisms and Model M6 equipped with a filter ensemble and ZeroFedAvg. Experiment 7 revealed that with a modest 2% poisoning, Model M5 managed an accuracy of 96.09%, while Model M6 fared better with 98.12%, suggesting the initial benefits of the defense strategies. The disparity became more pronounced in Experiment 8, where the poisoning increased to 4%; Model M5’s accuracy plummeted to 50.32%, whereas Model
Figure 18: Results showing the effectiveness of the filter ensemble and ZeroFedAvg against Data and Model Poisoning Attack

M6 showed remarkable resilience, maintaining an accuracy of 97.99%. Experiment 9 pushed the poisoning to 8% and continued the trend, with Model M5 dropping to an accuracy of 56.86% and Model M6 barely impacted at 97.97% accuracy. The results across the board highlight a secure FL environment, underscoring the filter ensemble and ZeroFedAvg’s effectiveness in mitigating the dual threat of data and model poisoning within a FL framework.
Chapter 7
Discussion and Conclusion

This thesis explored the efficacy of two defense mechanisms within a FL framework: a filter ensemble for data poisoning attacks and continuous verification of client updates for model poisoning attacks. This filter ensemble technique helped mitigate poisoned data before it could influence the learning process. On the other hand, continuous verification of client updates emerged as an effective strategy against model poisoning attacks. Our approach underscores the principle of zero trust within the FL context, where no participant is implicitly trusted. The continuous verification of client updates is implemented. However, since it heavily relies on the client modality values, the defense mechanism may fail to function effectively if a client sends incorrect or tampered modality values during the FL initialization process.

In Figure 19, all client modalities have been tampered by setting their values to 1. In this scenario, an attacker could create a poisoned model and encrypt it using the key 1. Since the server uses the same key for all modalities, the defense mechanism would fail to prevent the attack. This issue can be addressed by implementing strict data type and parameter validation for client modality values during the initialization phase. All other potential issues and future works are discussed below:

1. Carefully crafted data could launch targeted or untargeted attacks: Clients with carefully curated poisoned data may appear as seemingly real-looking adversaries, and the effects
of those can be observed after several server iterations. Rigorous statistical analysis of the parameter vectors of clients in the Central Aggregation Server (CAS) is needed, which is one area of future work.

2. Computation cost of filter ensemble: We have incorporated an additional model within the client for the filter ensemble. This additional model contains subcomponents that require training and testing, which increases the computational cost for the client. Therefore, it is crucial to design a lightweight Ensemble of Classifier (EOCL) model for the filter ensemble. To enhance computational efficiency, the development of zero-shot or few-shot EOCL algorithms could be pursued in future work.

3. Computation cost of global model: FL is a bi-level programming problem where the upper level involves a single minimization problem, and the lower level encompasses several minimization problems. It is inherently an NP-hard problem. Adding an extra component within the client introduces additional constraints at the lower level, thereby increasing the cost of the global model. We can add constraints to create a smaller search space for both the upper and lower layers, thereby narrowing the scope of the problem. This is an area for future work.

In conclusion, all scenarios were rigorously tested for data and model poisoning attacks, both with and without the implementation of the filter ensemble and continuous verification of client updates. The findings conclusively demonstrate that the proposed solution significantly enhances the security and robustness of the FL framework against such threats. It ensures a more secure and privacy-preserving environment for FL applications, reinforcing the overall integrity of the system.
References

URL https://market.us/report/federated-learning-market/

URL https://www.grandviewresearch.com/industry-analysis/federated-learning-market-report


URL https://gdpr-info.eu/

URL https://oag.ca.gov/privacy/ccpa


URL https://www.mdpi.com/2076-3417/12/2/734

URL https://doi.org/10.1145/3383455.3422562


[34] J. Liu, J. Huang, Y. Zhou, X. Li, S. Ji, H. Xiong, D. Dou, From distributed machine learning to federated learning: a survey, Knowledge and Information Systems 64 (4) (2022) 885–917. doi:10.1007/s10115-022-01664-x. \url{http://dx.doi.org/10.1007/s10115-022-01664-x}


[56] D. Lie, Minimizing the TCB. URL https://www.usenix.org/conference/14th-usenix-security-symposium/minimizing-tcb


54


