# Secure and Resilient Federated Learning

# Secure and Resilient Federated Learning

**Dissertation**

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on
[date= weekday (word) day (number), month (word) year (number)] at [hh:mm (number)] o'clock

by

**Rui WANG**

Master of Science in Cyber Security,
University of Southampton, the United Kingdom,
born in Kaili, China

This dissertation has been approved by the promotors.

*Keywords:* Federated Learning, Privacy-preservation, Byzantine Robustness

*Printed by:*

*Cover by:* Rui Wang

ISBN 000-00-0000-000-0

An electronic copy of this dissertation is available at
https://repository.tudelft.nl/.

To my beloved family.

# Contents

# SUMMARY

Summary in English...

# SAMENVATTING

Samenvatting in het Nederlands…

# PREFACE

Preface goes here. This chapter is optional.

This is the TU Delft dissertation template for LaTeX. The source files can be found at GitLab[1]. It is designed to work with all versions of LaTeX. However, if you want to strictly adhere to the TU Delft corporate design style[2], you need to use XeLaTeX, as it supports TrueType and OpenType fonts. This is needed to get Arial working.

This Overleaf template only supports the combination of Arial[3] with Roboto Slab[4] or Roboto[5] with Roboto Slab. The GitLab project mentioned earlier supports the font Utopia[6] using the LaTeX package Fourier[7] as well, and has many more features which were removed here for the sake of simplicity.

*Rui Wang*
*Delft, January 2013*

1 https://gitlab.com/novanext/tudelft-dissertation
2 https://www.tudelft.nl/en/tu-delft-corporate-design
3 https://en.wikipedia.org/wiki/Arial
4 https://fonts.google.com/specimen/Roboto+Slab
5 https://fonts.google.com/specimen/Roboto
6 https://fonts.adobe.com/fonts/utopia
7 https://ctan.org/pkg/fourier

# 1 INTRODUCTION

Federated Learning (FL) [1] is a paradigm of machine learning that enables multiple parties to collaboratively train a model without sharing their private data. It was first proposed by Google in 2016 [1] as a way to leverage the massive amount of data generated by edge devices while preserving user privacy and reducing communication costs. Since then, FL has attracted increasing attention from both academia and industry, as it offers a promising solution to access various domains such as healthcare [2, 3], finance [4], the Internet of Things [5], and edge computing [6].

Despite its benefits, FL still has privacy issues since clients' sensitive information may be leaked through the model updates exchanged during the training process [7–9]. Meanwhile, security issues are also notable. A fraction of clients may behave unpredictably or maliciously during the training process to deteriorate the accuracy of the global model [10] or inject a backdoor into the global model [11, 12].

A lot of works have been dedicated to these issues but they still pose new technical challenges that require innovative methods and tools to address. Among these challenges, the practicality of privacy-preserving horizontal & vertical FL and improving the Byzantine robustness of FL (Byzantine-robust FL aims to proceed with training properly even in the existence of malicious clients) are considered in this thesis for the following reasons. Firstly, privacy-preserving horizontal FL exposes many problems in terms of practicality. For example, encryption-based solutions rely on a Trusted Third Party (TTP) to distribute keys and involve frequent communication between a central server and multiple distributed clients, which can be costly and unreliable. Beyond that, it introduces a significant computational burden on FL. Secondly, in privacy-preserving vertical FL, the assumption that only one client has all the labels for all the samples may not be practical in some cases. For example, in a healthcare scenario, different hospitals may have diagnoses for different patients, and there may be inconsistent labels. Thirdly, most Byzantine-robust FL systems rely on the assumption of an honest majority of clients. In real-world scenarios, FL is often deployed in environments where there may be competing interests, and clients may have incentives to manipulate the learning process. Acknowledging the possibility of a malicious majority of clients cannot be ignored. Overall, it can be said that all three issues above must be resolved before bringing FL into real-world scenarios. This chapter is an introduction to the concept of FL that forms the basis of the thesis. In Section 1.1, we explain the FL and its relevance with distributed learning. Section 1.2 identifies the key challenges in the current state-of-the-art of FL, which are addressed in this thesis. The problem statement and research questions of this thesis are presented in Section 1.3. Finally, Section 1.4 lists our contributions and the outline of this thesis.

## 1.1. Federated Learning

Federated Learning (FL) [1] is a specialized paradigm within machine learning that pertains to the training of algorithms, such as deep learning, across numerous distinct local datasets residing within individual clients. Notably, this approach circumvents the explicit exchange of training datasets among clients. The fundamental tenet of FL involves the iterative training of local models on datasets owned by different clients, coupled with periodic updates exchanged with a central server. This exchange encompasses crucial model updates, such as the parameters or gradients inherent to a deep neural network. At defined intervals, the server aggregates updates to derive a global model shared ubiquitously across all participating clients.

In contrast, distributed learning [13], while sharing similarities with FL in terms of training a global model across multiple clients, diverges in its fundamental assumptions concerning the local datasets. The focus of distributed learning is to divide and distribute computing tasks on different devices so that they can be completed in parallel. It assumes that local datasets are Independent and Identical Distribution (IID) and that they have homogeneous data distributions. Unlike the rigorous assumptions inherent to distributed learning, data distributions in FL are more heterogeneous, namely non-IID. Figure 1.1 provides an illustration of IID and non-IID datasets with MNIST [14]. Specifically, in the context of IID, each client possesses all classes of labels, with an equal quantity of each class. In contrast, in non-IID scenarios, the quantity of labels distributed varies among clients, leading to significant distribution disparities. In an extreme case, as depicted in Figure 1.1, each client only holds 2 out of 10 labels. Overall, the constituent datasets within FL are typified by their dissimilarity and can exhibit substantial disparagement in their quantities of each class. Furthermore, the clients participating in FL may be inherently less reliable, owing to their susceptibility to failures and dropout events. This fragility comes from their reliance on weaker communication methods, exemplified by Wi-Fi, and battery-powered frameworks, like smartphones and IoT devices. In contrast, in the realm of distributed learning, clients predominantly comprise data centers endowed with abundant computational resources and are interlinked via high-speed networks.

### Definition

Empirical Risk Minimization (ERM) [15] is a fundamental concept in machine learning. It involves minimizing the empirical risk, also called loss, on a training dataset by optimizing model parameters. Considering the paradigm of ERM within FL, the primary objective is to acquire knowledge encapsulated within a global model denoted as $f_\theta : \mathcal{X} \to \mathcal{Y}$, which serves to map a given data instance $x \in \mathcal{X}$ to an associated label $y \in \mathcal{Y}$. Note that the FL server is intentionally precluded from directly accessing the underlying training dataset. Instead, it orchestrates a process of aggregating local updates (weights or gradients), collected from local clients, each of which undertakes autonomous training on their respective local datasets.

The prevalent operational approach for synthesizing these updates is commonly termed as `Federated Averaging` (`FedAvg`) [1]. The overarching training objective revolves around the attainment of the global parameters $\theta$ by means of addressing

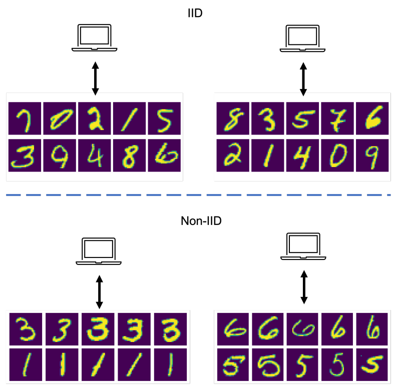


Figure 1.1: An illustration of IID and non-IID for MNIST dataset, where the distributions of the quantity of labels are more evenly in the case of IID compared to the cause of non-IID.

the finite-sum optimization challenge formulated as follows:

$$\underset{\theta}{\operatorname{argmin}}\ f_\theta = \frac{1}{n}\sum_{i=1}^{n} f_{\theta_i}, \tag{1.1}$$

where $n$ signifies the count of participating clients. Under FL setting, each client $i$ holds its training data $\mathscr{X}_i$ and labels $\mathscr{Y}_i$. Within each round indexed by $t$, the server $S$ identifies $n$ clients for participation in the update aggregation endeavor. Then the server broadcasts the global model $\theta^t$ to the selected clients. Subsequently, each client $i$ proceeds to train an independent local model $f_{\theta_i} : \mathscr{X}_i \to \mathscr{Y}_i$, employing its localized dataset $D_i = \{(x_j, y_j) : x_j \in \mathscr{X}_i, y_j \in \mathscr{Y}_i, j = 1, 2, ..., |D_i|\}$ for a designated number of training epochs. This training is executed via an optimization algorithm, most commonly Stochastic Gradient Descent (SGD) [16].

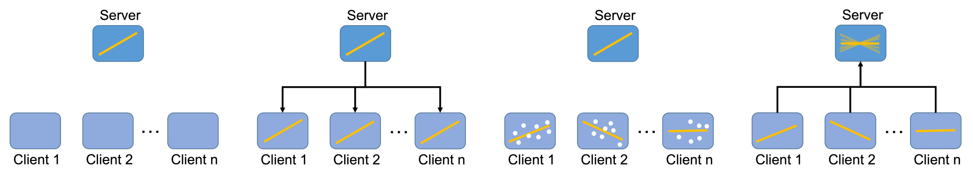
The main goal of client $i$ is to develop a refined local model, captured by the optimization task as expressed below:

$$\theta_i^* = \underset{\theta^t}{\operatorname{argmin}} \sum_{(x_j, y_j)\in D_i} \mathscr{L}(f_{\theta^t}(x_j), y_j). \tag{1.2}$$

Within this expression, $\mathscr{L}$ symbolizes the employed loss function, often exemplified by the cross-entropy loss in the classification task. Subsequent to this optimization, client $i$ generates local updates denoted as $\delta_i^t = \theta_i^* - \theta^t$, which is subsequently sent back to the server. Finally, the server aggregates the diverse updates and forges an updated global model via averaging, which is succinctly represented as:

$$\theta^{t+1} = \theta^t + \frac{\eta}{n}\sum_{i\in[n]} \delta_i^t. \tag{1.3}$$

Herein, $\eta$ embodies the global learning rate. Once the global model $\theta$ achieves convergence or the training protocol reaches a prescribed iteration limit, the aggregation process is terminated, yielding the final global model. The overview of the training process is summarized in Figure 1.2.



(a) The Server selects $n$ clients for FL training.

(b) The Server sends the initial model to the clients.

(c) Clients conduct local training with local datasets.

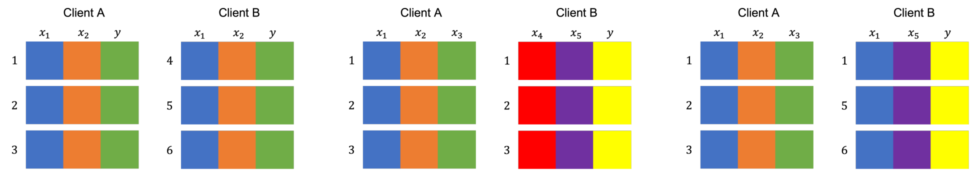
(d) Clients upload updates aggregated by the server.

Figure 1.2: Overview of Federated Learning training process. The boxes stand for participants of FL. The yellow line and white points represent model and training data, respectively. The model fits the data as much as possible to achieve the minimal loss.

## Categories of Federated Learning

In this section, we explore the categorization of FL, focusing on the distribution characteristics of the data.

Given that the feature ($\mathcal{X}$), label ($\mathcal{Y}$), and sample ID ($I$) space of data owned by clients may differ, the FL can be categorized into three types: horizontal FL, vertical FL, and Federated Transfer Learning (FTL). These classifications are based on how data is distributed among various clients in both the feature and sample ID space. Figure 1.3 illustrates the different frameworks for a two-party scenario in FL.



(a) Horizontal Federated Learning

(b) Vertical Federated Learning

(c) Federated Transfer Learning

Figure 1.3: Categorization of Federated Learning. The $x_i$ and $y$ stand for different features and labels respectively, showing as different colors. The numbers on the left side of the features represent sample IDs.

- Horizontal FL: In this category, different clients possess the same features but have different sample IDs of data, which can be summarized as:

$$\mathcal{X}_i = \mathcal{X}_j, \mathcal{Y}_i = \mathcal{Y}_j, I_i \neq I_j, \forall D_i, D_j, i \neq j. \tag{1.4}$$

Each client trains its local model on its unique subset of data. Horizontal FL is suitable when the overall goal is to aggregate knowledge about a common set of features across decentralized data sources.

- Vertical FL: It is employed when clients have different features, but there is an overlap or correlation between them. Each client holds specific information about a subset of features, and the FL process involves collaboratively training a model across these complementary feature sets. This type is particularly useful in scenarios where combining diverse data sources enhances the model's performance. Therefore, the vertical FL is defined as:

$$\mathcal{X}_i \neq \mathcal{X}_j, \mathcal{Y}_i \neq \mathcal{Y}_j, I_i = I_j, \forall D_i, D_j, i \neq j. \tag{1.5}$$

- Federated Transfer Learning: It involves the scenarios in which the two data sets differ not only in sample IDs but also in feature space. We have:

$$\mathcal{X}_i \neq \mathcal{X}_j, \mathcal{Y}_i \neq \mathcal{Y}_j, I_i \neq I_j, \forall D_i, D_j, i \neq j. \tag{1.6}$$

  The goal is to train a model on a source domain and then adapt it to perform well on a target domain. This is especially valuable when there is a scarcity of labeled data in the target domain. The model is initially trained on one or more source domains, and the knowledge gained is transferred and fine-tuned on the target domain while respecting data privacy constraints.

These categories provide a framework for understanding how FL can be applied in various scenarios, depending on the nature of the data distribution and the specific goals of the learning process. The choice of category depends on factors such as the similarity of features across clients, the availability of labeled data, and the desired outcome of the FL tasks.

### Unsolved Privacy Issues within FL

The goal of proposing FL is to preserve the privacy of the training data of clients by allowing models to be trained across local clients without sharing raw data. However, it's essential to recognize that while FL enhances privacy compared to traditional centralized approaches, it may not provide absolute privacy guarantees.

Within the domain of FL, adversaries can engage in activities like parameter analysis or gradient-matching attacks, endeavoring to reverse-engineer private data from seemingly harmless updates [8, 17–19]. This risk highlights the importance of using methods that protect privacy. Therefore, it's crucial to be watchful in safeguarding not just the data itself but also the metadata and additional information that might accidentally expose sensitive details, like model updates. For example, deep leakage from gradients [8] is a privacy attack that can recover the private training data from the publicly shared gradients in FL. Specifically, the attack is based on the assumption that the gradients are computed from a single data point or a small batch of data points and that the attacker has access to the model architecture and parameters. The basic idea is to use the gradients as guidance to iteratively update a dummy data sample until it converges to the raw data. The attack works as follows: 1) The attacker generates a dummy data sample and a target label with the same shape as the training data and then derives the dummy gradients. 2) The attacker computes the Euclidean distance between clients' gradients and dummy gradients as a reconstruction loss. 3) The attacker updates the dummy data by applying the gradient descent rule. 4) The attacker repeats steps 2) and 3) until the loss is minimized or the dummy data stops changing significantly. 5) The attacker obtains the recovered data sample, which is close to the training data point in terms of pixel-wise or token-wise precision.

This attack is much stronger than the approaches that can only infer some statistical information or membership information from the gradients [20]. The attack poses a serious threat to the privacy of the clients in the FL system.

Therefore, one of the paramount concerns revolves around preserving privacy, a challenge that manifests itself in multifaceted dimensions. To address these privacy concerns in FL, privacy-preserving algorithms are essential. However, the FL setting imposes unique constraints on these algorithms. Firstly, they must exhibit practicality and efficiency, avoiding dependence on unrealistic assumptions and excessive computational and communication burdens on the FL system. Secondly, these methods must do so without unduly sacrificing the accuracy of the global model, as FL's efficacy relies on the quality of the model learned.

### Unsolved Robust Issues within FL

Byzantine robustness within the context of FL becomes a critical concern. As FL uses the combined knowledge of many decentralized clients to train a shared model, the risk of some clients trying to take advantage of the cooperative system for malicious purposes becomes clear [21–24]. Such clients, also known as Byzantine clients, endeavor to subvert the learning process by employing a diverse array of tactics (i.e., poisoning attacks), thereby posing multifaceted threats to the overall integrity, security, and efficacy of the FL system.

Untargeted attacks [10, 25], emblematic of the first facet of this issue, involve Byzantine clients engaging in disruptive behavior with the primary aim of causing harm to the accuracy and performance of the FL model, without having a specific objective or target in mind. These tactics encompass uploading incoherent, spurious, or corrupted local updates to the server, thereby introducing significant discord into the training procedure. Furthermore, such adversarial entities may deliberately introduce perturbations or anomalies to their local datasets, thereby propagating noise and divergence into the global model [25]. Consequently, the outcomes of untargeted attacks manifest in the degradation of the global model's accuracy, a perturbation in its stability, and a hindered convergence rate, thereby hampering the overall efficacy of the FL ecosystem.

The second facet of this problem is represented by targeted attacks [11, 12, 26–28], where Byzantine clients adopt a purposeful approach by aiming their adversarial actions toward specific objectives. These malicious objectives often involve the introduction of maliciously crafted local updates into the global model, leading to the injection of erroneous patterns or the creation of subtle triggers, often referred to as backdoors [28]. Additionally, manipulation of the local dataset to carry hidden vulnerabilities or exploit points further exacerbates the security compromise.

## 1.2. CHALLENGES IN SECURITY OF FEDERATED LEARNING

Given the expansive scope of the field of FL, it becomes evident that a great number of challenges necessitate resolutions to push the current state-of-the-art in FL towards greater advancement. In this dissertation, we take the first step to enhance the security of FL in various real-world situations. This section succinctly enumerates the predominant challenges in the realm of secure FL. We will delve into the intricate challenges entailing matters of the practicality of privacy-preserving horizontal & vertical FL and improving the Byzantine robustness of FL. Nonetheless, the non-inclusion of challenges, such as fairness considerations within FL, within the scope of this study does not imply their significance is diminished.

### PRACTICALITY OF PRIVACY-PRESERVING HORIZONTAL FL

Differential Privacy (DP) [29] and Homomorphic Encryption (HE) [30–33] have been incorporated into FL as supplementary privacy-preserving mechanisms. DP introduces Gaussian [34] or Laplacian [35] noise to each uploaded model gradient [36, 37], a cost-effective and lightweight approach. Nevertheless, the introduction of noise adversely affects model performance in terms of accuracy and does not address potential data reconstruction attacks on the model gradients [38].

HE also plays a pivotal role in the preservation of privacy within the landscape of FL since it allows the server to do aggregations under ciphertext. Specifically, the process involves encrypting the model parameters or updates on the client side before uploading to the server. The server can perform aggregations on ciphertext without decrypting it, obtaining the encrypted results. Once the server transmits the encrypted result back to the clients, they can decrypt it to reveal the final model update. This entire process ensures that the server never accesses the raw data, preserving the privacy of individual client contributions throughout the FL process. However, the HE-based FL [39–41] is marked by a salient drawback that requires close attention. This drawback arises because it necessitates the participation of a Trusted Third Party (TTP) in the crucial tasks of creation and distribution of cryptographic key pairs. While HE plays a crucial role in privacy-preserving FL, adding a TTP brings a complex set of challenges that may impact FL frameworks in terms of efficiency and security.

Primarily, incorporating a TTP increases the attack surface and overall complexity of the topology within the FL system. The involvement of an additional entity, responsible for key management and distribution, introduces an inherent hierarchical structure that deviates from the decentralized essence characterizing FL systems. This departure from the fundamental tenets of FL can lead to intricate network configurations. Consequently, the incorporation of a TTP in HE-based FL systems brings about additional communication complexity, heightened communication costs, and the potential for bottlenecks during key distribution. These factors collectively impact the efficiency and scalability of the FL process, potentially diminishing the advantages offered by HE for secure and privacy-preserving computations.

Furthermore, in the context of HE-based FL, a notable distinction arises when compared to other areas such as banking that inherently operate with TTP. Unlike these domains, HE-based FL faces a unique challenge  the need to introduce a new TTP. In traditional sectors like banking, the TTP is an integral part of the system, and the ecosystem itself functions as a TTP. However, in HE-based FL, establishing trust becomes even more critical. In the event of a compromise of the TTP, whether through external breaches or insider collusion, the integrity of the cryptographic keys is easily compromised. This further exposes the confidential model updates and aggregated knowledge shared by participating clients to potential interception and manipulation by attackers. The gravity of such an occurrence is particularly pronounced in scenarios where FL is deployed in security-sensitive domains such as medical research, financial analysis, and industrial innovation, where privacy and integrity of data are paramount.

Beyond that, cryptographic methodologies [42–45] substantively enhance both the confidentiality and integrity of the model updates as well as the locally held data, thereby finding frequent application within the realm of FL. However, the incorporation of HE concurrently brings in a set of challenges with regard to the optimization of communication and computation procedures. This arises due to the inherent need for supplementary computational and communicative steps involved in encryption, decryption, and aggregation tasks inherent to cryptographic computations.

One illustrative example pertains to HE, which facilitates arithmetic operations on encrypted model updates without necessitating their prior decryption. Nevertheless, the utilization of HE in FL induces an augmentation in terms of computational complexity [41]. This enhancement in complexity can lead to escalated computational overheads, as the mathematical operations performed over ciphertext inherently demand a more substantial amount of computational resources than their plaintext counterparts. Consequently, the increased computational load contributes to taking a longer time to complete training, which can ultimately lead to a severe slowdown in the FL process [38].

Furthermore, the utilization of HE in the context of FL introduces an urgent need for elevated communication resources [41]. As the encryption and decryption procedures are executed on ciphertexts, the requisite exchange of encrypted model updates between clients and the server leads to an enlargement in the volume of data transmitted through the communication channels. This inflation in the data size can impede the efficiency of data transfer, leading to not only delays in transmitting updates but also an increase in the overall communication overheads. This is particularly pronounced in scenarios where network bandwidth is constrained, as the data transmission rate may become a limiting factor in the FL workflow.

### Practicality of Privacy-preserving Vertical FL

Another problem in the practicality of privacy-preserving FL is label distribution for Vertical Federated Learning (VFL) [4, 46–50], where features of training data are vertically partitioned. The prevailing methodologies presuppose a sole client's control over the management and processing of all training labels. However, practical cases underscore the necessity for a VFL scheme that accommodates the distributed nature of these labels.

To illustrate, consider the context where multiple medical establishments such as hospitals, clinics, and health centers are designated as COVID-19 testing facilities. Their collective objective is to collaboratively train a predictive model. This model seeks to provide informed predictions about the infection status of individuals in different geographic locations, relying on their personal health records and symptoms.

In this real-world scenario, the labels denoting specific outcomes, such as the test results, are apt to be spread among distinct health authorities. Moreover, the feature space exhibits vertical partitioning, meaning that each authority retains dominion over a subset of attributes. For example, a specialized cardiac hospital exclusively maintains cardiac-related data, whereas a psychiatric center exclusively houses mental health records. Each authority tends to collect and manage the labels associated with their registered patients locally.

Another illustrative context pertains to the financial sector, where diverse branches of banks and e-commerce enterprises seek to build a global predictive model. This model aims to predict the likelihood of timely payments for certain services, such as automobile loans, extended to their customers. The banks contribute a subset of customer-related features, encompassing metrics like account balances and transaction history. Conversely, the e-commerce entities augment the feature pool with data pertaining to payment preferences and behaviors. Given that customers may avail of the same services from different authorities, the importance of assumption of distributed labels, rather than centralized ones, becomes self-evident.

## Improving Byzantine robustness of FL

Defending against malicious clients [10–12, 25–28] is the main goal of Byzantine-robust FL. The predominant paradigms in Byzantine-robust FL [21, 23, 51], primarily depend on the fundamental assumption of a majority of honest clients. This basic idea posits that the proportion of Byzantine clients is limited, ensuring a scenario wherein a substantial portion of participating clients can be relied upon to provide genuine and reliable updates. These conventional mechanisms are meticulously designed to perform well under these ideal circumstances.

To clarify this further, the Krum and Multi-Krum algorithms [10] serve as examples of such methodologies. These algorithms follow a philosophy based on finding the most similar local updates. They perform malicious identifications by using the distance between pair-wise updates. Subsequently, they exclude outliers that deviate significantly from all the rest of the updates. In a parallel vein, the coordinate-wise median algorithm [10] adopts a per-coordinate approach to computing the median value of local updates, thereby offering protection against the undue influence of extreme values.

While these frameworks have proven to be effective within the realms of conventional FL, they do exhibit vulnerabilities when confronted with the reality of a malicious majority of clients. If more than half of the participating clients manifest Byzantine behavior, the entire FL system stands at the precipice of a significant crisis. In such a scenario, the server becomes considerably challenged in its ability to discriminate between benign and malicious updates. Consequently, the final global model is highly compromised to the point of being inaccurate. These factors collectively underscore the urgent needs for the formulation and deployment of novel strategies in the domain of Byzantine-robust FL.

## 1.3. Problem Statement

FL has been evolving in the interdisciplinary research community of machine learning, security, and distributed systems. This thesis aims to address highly impactful problems regarding circumventing the dependency on TTP for HE-based FL, improving the efficiency of HE-based FL, decentralized labels in vertical FL, and defending against the malicious majority of clients. Here, we present the research question of the thesis, which can be explained in two parts: the practicality of privacy-preserving FL and the improving Byzantine robustness of FL.

### Practicality of Privacy-preserving FL

The practicality of privacy-preserving FL refers to deploying FL in real-world scenarios without leaking clients' privacy. The current paradigms based on HE, while effective in preserving privacy, introduce a TTP. Therefore, their complexities compromise the decentralization, efficiency, and security of FL, which brings the first research question:

**Q1:** *What developments and considerations are involved in designing a homomorphic encryption-based federated learning system without relying on a trusted third party?*

Another problem in the practicality of privacy-preserving FL is the computational and communication overheads of integration of cryptographic methodologies, particularly HE, leading to potential bottlenecks in FL processes. Specifically, the exchange of encrypted model updates introduces heightened communication requirements, impacting data transfer efficiency. Addressing these challenges is crucial for optimizing the performance of FL systems, ensuring the delicate balance between privacy preservation and computational efficiency in cryptographic calculations. Regarding these, we present our second research question:

**Q2:** *How to optimize cryptographic calculations to reduce computational and communication overheads?*

Apart from the horizontal FL, existing privacy-preserving vertical FL methods still have a significant gap from being applied in the real world. The identified problem revolves around the decentralized nature of labels in vertically partitioned datasets, a scenario often overlooked by current approaches. Illustrative contexts in healthcare and finance underscore the practicality and urgency of developing

VFL schemes capable of accommodating such label distribution. The existing works are limited to centralized labels in VFL, which brings the third research question:

**Q3:** *How to preserve the privacy of clients in the vertical federated learning system if the labels are decentralized?*

### IMPROVING BYZANTINE ROBUSTNESS OF FL

The challenge of achieving Byzantine robustness in Federated Learning becomes particularly pronounced when the majority of participating clients exhibit malicious behavior. While existing methodologies like Krum and Multi-Krum demonstrate effectiveness under the assumption of a majority of honest clients, their vulnerability in the face of a malicious majority is evident. The current paradigms struggle to distinguish between authentic and malicious updates in such scenarios, leading to compromised global models with significant inaccuracies. Addressing this issue is paramount for the advancement of Byzantine-robust Federated Learning, which brings our fourth question:

**Q4:** *How to capture the Byzantine robustness of honest clients in federated learning if the majority of clients are malicious?*

## 1.4. CONTRIBUTION OF THE THESIS

The thesis is structured such that each technical chapter comprises a self-contained replication of a research paper. These chapters are designed to be autonomously comprehensible, allowing for standalone reading. We have endeavored to retain the technical details of the original publications, albeit with possible minor modifications. Consequently, readers may encounter various notations, overlapping introductions, and literature review sections across different chapters. The outline of the thesis is given as follows:

### CHAPTER 2

### DISTRIBUTED ENCRYPTION FOR PRIVACY-PRESERVING FEDERATED LEARNING

In this chapter, we address the research questions **Q1** and **Q2** and present our contributions to the practicality of privacy-preserving horizontal FL regarding getting rid of TTP and improving computational and communication efficiency. Firstly, we introduce an efficient threshold encryption framework featuring federated key generation and model quantization, tailored for deployment within federated deep learning systems. This innovation results in notable reductions in the number of model parameters requiring encryption and the communication overheads associated with transmitting local models. Furthermore, the need for an additional TTP is rendered obsolete. Secondly, to facilitate the secure aggregation of model updates, an approximate aggregation method is devised for the server. This method segregates the uploaded ciphertexts and quantized gradients, thereby enabling the server to transmit aggregated ciphertexts to only those clients who meet a predefined threshold value for distributed partial decryption. Consequently, our proposed approximate aggregation mechanism significantly diminishes the computational and communication costs incurred during the threshold decryption process. The chapter is an integral copy of the paper *"Distributed Additive Encryption and Quantization for Privacy Preserving Federated Deep Learning"* by Zhu, H., Wang, R., Jin, Y., Liang, K., and Ning, J. Neurocomputing, 463, pp.309-327.

## Chapter 3

### Secure Vertical Federated Learning for Decentralized Labels

In this chapter, we address the research question **Q3** that concerns the privacy breach when labels are decentralized in VFL. We have undertaken the task of defining VFL within a practical context, particularly when confronted with the distribution of training labels across multiple clients. In addition to this conceptual groundwork, we have introduced a novel approach for secure and efficient XGBoost training. This approach elegantly integrates secure aggregation techniques, which are rooted in the principles of Diffie-Hellman key exchange and key derivation function, as well as global differential privacy. Furthermore, we conducted an extensive security analysis of the designed protocol, which serves to illustrate its efficacy in preserving the confidentiality of label values and feature data privacy within the semi-honest setting. Notably, it exhibits resilience, even in scenarios where collusion is exhibited by $n-2$ out of $n$ clients. The chapter is an integral copy of the paper *"FEVERLESS: Fast and Secure Vertical Federated Learning based on XGBoost for Decentralized Labels"* by Wang, R., Ersoy, O., Zhu, H., Jin, Y. and Liang, K. IEEE Transactions on Big Data.

## Chapter 4

### Taming Malicious Majority of Clients in Federated Learning

In this chapter, we address the research question **Q4**, and we propose the first Byzantine-robust FL system that can defend against the malicious majority of clients. We present a novel aggregation strategy designed to enhance the Byzantine robustness of FL by effectively mitigating poisoning attacks originating from a majority of malicious clients, all while obviating the necessity for servers to possess an auxiliary dataset. This approach asserts that the deployment of intricate algorithms for the detection of malicious updates is redundant. Instead, it advocates implementing measures aimed at preventing the coexistence of malicious and semi-honest clients within a single aggregation. We introduce a novel technique tailored to augment the accuracy of updates clustering, particularly in non-IID scenarios. Instead of directly employing the updates for clustering, we adopt a two-step process. Initially, we calculate the pairwise adjusted cosine similarity of updates, taking into account dissimilarity in both directions and magnitudes between updates from any two clients. Subsequently, these results are input into the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm. Furthermore, we verify the security claims for the framework within the universal composability framework, offering formal security proof. This proof accommodates dynamic security requirements, rendering the protocol not only theoretically sound but also practically robust.

### 1.4.1. List of Excluded Publications

In the following, we list the papers published during Ph.D. but not included in the thesis since they only present partial elements of the chapters.

- Tjiam, K., **Wang, R.**, Chen, H. and Liang, K., 2021, November. Your smart contracts are not secure: investigating arbitrageurs and oracle manipulators in Ethereum. In Proceedings of the 3rd Workshop on Cyber-Security Arms Race (pp. 25-35).
- Sun, L., Du, R., He, D., Zhu, S., **Wang, R.** and Chan, S., 2021, December. Feature Engineering Framework based on Secure Multi-Party Computation in Federated Learning. In 2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys) (pp. 487-494). IEEE.
- Zhu, H., **Wang, R.**, Jin, Y. and Liang, K., 2021. Pivodl: Privacy-preserving vertical federated learning over distributed labels. IEEE Transactions on Artificial Intelligence.
- Liu, A., Li, B., Li, T., Zhou, P. and **Wang, R.**, 2022. AN-GCN: An Anonymous Graph Convolutional Network Against Edge-Perturbing Attacks. IEEE transactions on neural networks and learning systems.
- Tian, Y., **Wang, R.**, Qiao, Y., Panaousis, E. and Liang, K., 2023, April. FLVoogd: Robust And Privacy Preserving Federated Learning. In Asian Conference on Machine Learning (pp. 1022-1037). PMLR.
- Xu, J., **Wang, R.**, Koffas, S., Liang, K. and Picek, S., 2022, December. More is better (mostly): On the backdoor attacks in federated graph neural networks. In Proceedings of the 38th Annual Computer Security Applications Conference (pp. 684-698).
- Ghavamipour, A.R., Turkmen, F., **Wang, R.** and Liang, K., 2023, May. Federated Synthetic Data Generation with Stronger Security Guarantees. In Proceedings of the 28th ACM Symposium on Access Control Models and Technologies (pp. 31-42).

# REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y Arcas. 'Communication-efficient learning of deep networks from decentralized data'. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.

[2] I. Dayan, H. R. Roth, A. Zhong, A. Harouni, A. Gentili, A. Z. Abidin, A. Liu, A. B. Costa, B. J. Wood, C.-S. Tsai *et al.* 'Federated learning for predicting clinical outcomes in patients with COVID-19'. In: *Nature medicine* (2021), pp. 1735–1743.

[3] N. Rieke, J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein *et al.* 'The future of digital health with federated learning'. In: *NPJ digital medicine* 3.1 (2020), p. 119.

[4] Y. Wu, S. Cai, X. Xiao, G. Chen and B. C. Ooi. 'Privacy Preserving Vertical Federated Learning for Tree-Based Models'. In: *Proc. VLDB Endow.* (2020), pp. 2090–2103.

[5] O. Marfoq, G. Neglia, A. Bellet, L. Kameni and R. Vidal. 'Federated Multi-Task Learning under a Mixture of Distributions'. In: *Advances in Neural Information Processing Systems*. 2021, pp. 15434–15447.

[6] Z. Zhu, J. Hong, S. Drew and J. Zhou. 'Resilient and Communication Efficient Learning for Heterogeneous Federated Systems'. In: *Proceedings of the 39th International Conference on Machine Learning*. Proceedings of Machine Learning Research. 17–23 Jul 2022, pp. 27504–27526.

[7] T. Orekondy, S. J. Oh, Y. Zhang, B. Schiele and M. Fritz. 'Gradient-Leaks: Understanding and Controlling Deanonymization in Federated Learning'. In: *arXiv preprint arXiv:1805.05838* (2018).

[8] L. Zhu, Z. Liu and S. Han. *Deep Leakage from Gradients*. 2019. arXiv: `1906.08935 [cs.LG]`.

[9] B. Hitaj, G. Ateniese and F. Perez-Cruz. 'Deep models under the GAN: information leakage from collaborative deep learning'. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 603–618.

[10] M. Fang, X. Cao, J. Jia and N. Gong. 'Local Model Poisoning Attacks to {Byzantine-Robust} Federated Learning'. In: *USENIX Security*. 2020, pp. 1605–1622.

[11] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee and D. Papailiopoulos. 'Attack of the Tails: Yes, You Really Can Backdoor Federated Learning'. In: *NIPS*. 2020.

[12] C. Xie, K. Huang, P.-Y. Chen and B. Li. 'DBA: Distributed Backdoor Attacks against Federated Learning'. In: *ICLR*. 2020.

[13] J. Konen, B. McMahan and D. Ramage. 'Federated optimization: Distributed optimization beyond the datacenter'. In: *arXiv preprint arXiv:1511.03575* (2015).

[14] Y. LeCun and C. Cortes. 'MNIST handwritten digit database'. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[15] V. Vapnik. 'Principles of risk minimization for learning theory'. In: *Advances in neural information processing systems* 4 (1991).

[16] L. Bottou and O. Bousquet. 'The tradeoffs of large scale learning'. In: *Advances in neural information processing systems* 20 (2007).

[17] J. Geiping, H. Bauermeister, H. Dröge and M. Moeller. *Inverting Gradients – How easy is it to break privacy in federated learning?* 2020. arXiv: 2003.14053 [cs.CV].

[18] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz and P. Molchanov. 'See through gradients: Image batch recovery via gradinversion'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 16337–16346.

[19] Y. Huang, S. Gupta, Z. Song, K. Li and S. Arora. 'Evaluating gradient inversion attacks and defenses in federated learning'. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 7232–7241.

[20] A. Pustozerova and R. Mayer. 'Information leaks in federated learning'. In: *Proceedings of the Network and Distributed System Security Symposium*. Vol. 10. 2020, p. 122.

[21] P. Blanchard, E. M. El Mhamdi, R. Guerraoui and J. Stainer. 'Machine learning with adversaries: Byzantine tolerant gradient descent'. In: *NIPS*. 2017, pp. 118–128.

[22] Y. Chen, L. Su and J. Xu. 'Distributed statistical machine learning in adversarial settings: Byzantine gradient descent'. In: *POMACS* (2017), pp. 1–25.

[23] J. Xu, S.-L. Huang, L. Song and T. Lan. *SignGuard: Byzantine-robust Federated Learning through Collaborative Malicious Gradient Filtering*. 2021. arXiv: 2109.05872 [cs.LG].

[24] X. Cao, M. Fang, J. Liu and N. Z. Gong. 'FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping'. In: *NDSS*. 2021.

[25] T. D. Nguyen, P. Rieger, M. Miettinen and A.-R. Sadeghi. 'Poisoning attacks on federated learning-based IoT intrusion detection system'. In: *DISS*. 2020, pp. 1–7.

[26] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J. Sohn, K. Lee and D. S. Papailiopoulos. 'Attack of the Tails: Yes, You Really Can Backdoor Federated Learning'. In: *NIPS*. 2020.

[27] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin and V. Shmatikov. 'How to backdoor federated learning'. In: *AISTATS*. 2020, pp. 2938–2948.

[28] Z. Sun, P. Kairouz, A. T. Suresh and H. B. McMahan. 'Can you really backdoor federated learning?' In: *arXiv preprint arXiv:1911.07963* (2019).

[29] C. Dwork. 'Differential Privacy: A Survey of Results'. In: *Theory and Applications of Models of Computation*. Ed. by M. Agrawal, D. Du, Z. Duan and A. Li. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–19. ISBN: 978-3-540-79228-4.

[30] P. Paillier. 'Public-key cryptosystems based on composite degree residuosity classes'. In: *TAMC*. Springer. 1999.

[31] T. ElGamal. 'A public key cryptosystem and a signature scheme based on discrete logarithms'. In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[32] C. Gentry and D. Boneh. *A fully homomorphic encryption scheme*. Vol. 20. 9. Stanford University Stanford, 2009.

[33] I. Damgård, V. Pastro, N. Smart and S. Zakarias. 'Multiparty Computation from Somewhat Homomorphic Encryption'. In: *CRYPTO*. 2012.

[34] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar and L. Zhang. 'Deep learning with differential privacy'. In: *The 2016 ACM CCS*.

[35] R. Shokri and V. Shmatikov. 'Privacy-preserving deep learning'. In: *the 22nd ACM CCS*.

[36] R. C. Geyer, T. Klein and M. Nabi. 'Differentially private federated learning: A client level perspective'. In: *arXiv preprint arXiv:1712.07557* (2017).

[37] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek and H. V. Poor. 'Federated learning with differential privacy: Algorithms and performance analysis'. In: *IEEE Transactions on Information Forensics and Security* (2020).

[38] L. T. Phong, Y. Aono, T. Hayashi, L. Wang and S. Moriai. 'Privacy-Preserving Deep Learning via Additively Homomorphic Encryption'. In: *IEEE Transactions on Information Forensics and Security* 13.5 (2018), pp. 1333–1345. DOI: 10.1109/TIFS.2017.2787987.

[39] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang and Y. Zhou. 'A hybrid approach to privacy-preserving federated learning'. In: *The 12th ACM Workshop on AISec*.

[40] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar and H. Ludwig. 'HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning'. In: *The 12th ACM AISec*. 2019.

[41] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan and Y. Liu. 'Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning'. In: {*USENIX*} *ATC*. 2020, pp. 493–506.

[42] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl and N. P. Smart. 'Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits'. In: *ESORICS*. 2013, pp. 1–18.

[43] Y. Lindell and A. Nof. ‘A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority’. In: *CCS*. 2017, pp. 259–276.

[44] T. K. Frederiksen, M. Keller, E. Orsini and P. Scholl. ‘A unified approach to MPC with preprocessing using OT’. In: *ASIACRYPT*. 2015, pp. 711–735.

[45] I. Damgård, D. Escudero, T. Frederiksen, M. Keller, P. Scholl and N. Volgushev. ‘New primitives for actively-secure MPC over rings with applications to private machine learning’. In: *IEEE S&P*. 2019, pp. 1102–1120.

[46] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith and B. Thorne. ‘Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption’. In: *arXiv preprint arXiv:1711.10677* (2017).

[47] Y. Liu, X. Zhang and L. Wang. ‘Asymmetrically vertical federated learning’. In: *arXiv preprint arXiv:2004.07427* (2020).

[48] S. Yang, B. Ren, X. Zhou and L. Liu. ‘Parallel distributed logistic regression for vertical federated learning without third-party coordinator’. In: *arXiv preprint arXiv:1911.09824* (2019).

[49] Y. Liu, Y. Kang, L. Li, X. Zhang, Y. Cheng, T. Chen, M. Hong and Q. Yang. ‘A communication efficient vertical federated learning framework’. In: *Unknown Journal* (2019).

[50] X. Luo, Y. Wu, X. Xiao and B. C. Ooi. ‘Feature inference attack on model predictions in vertical federated learning’. In: *ICDE*. 2021, pp. 181–192.

[51] T. D. Nguyen, P. Rieger, H. Chen, H. Yalame, H. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, S. Zeitouni, F. Koushanfar, A.-R. Sadeghi and T. Schneider. ‘FLAME: Taming Backdoors in Federated Learning’. In: *USENIX Security*. 2022.

# 2
# Distributed Encryption for Privacy-Preserving Federated Learning

*Homomorphic encryption is a very useful gradient protection technique used in privacy-preserving federated learning. However, existing encrypted federated learning systems need a trusted third party to generate and distribute key pairs to connected participants, making them unsuited for federated learning and vulnerable to security risks. Moreover, encrypting all model parameters is computationally intensive, especially for large machine learning models such as deep neural networks. In order to mitigate these issues, we develop a practical, computationally efficient encryption based protocol for federated deep learning, where the key pairs are collaboratively generated without the help of a trusted third party. By quantization of the model parameters on the clients and an approximated aggregation on the server, the proposed method avoids encryption and decryption of the entire model. In addition, a threshold based secret sharing technique is designed so that no one can hold the global private key for decryption, while aggregated ciphertexts can be successfully decrypted by a threshold number of clients even if some clients are offline. Our experimental results confirm that the proposed method significantly reduces the communication costs and computational complexity compared to existing encrypted federated learning without compromising the performance and security.*

This chapter is based on the paper Distributed Additive Encryption and Quantization for Privacy Preserving Federated Deep Learning by Zhu, H., Wang, R., Jin, Y., Liang, K., and Ning, J. Neurocomputing, 463, pp.309-327.

## 2.1. Introduction

Federated learning (FL) [1] enables different clients to collaboratively train a global model by sending local model parameters or gradients to a server, instead of the raw data. Compared to traditional centralized learning, FL cannot only address the problem of isolated data island, but also play an important role in privacy preservation. Consequently, FL has been deployed in an increasing number of applications in mobile platforms, healthcare, and industrial engineering, among many others [2, 3].

However, FL consumes a considerable amount of communication resources. Model parameters or gradients need to be downloaded and uploaded frequently between the server and clients in each communication round, resulting in a longer operational stage especially for those large and complex models such as deep convolutional neural networks [4]. Much research work has been proposed to reduce the communication costs in the context of FL, including layer-wise asynchronous model update [5], reduction of the model complexity [6], optimal client sampling [7], and model quantization [8, 9] based on the trained ternary compression [10]. It has been empirically and theoretically shown that using quantization in FL does not cause severe model performance degradation [8, 11, 12], and under certain conditions, quantization can even reduce weight divergence [9].

Although FL can preserve data privacy to a certain degree, recent studies [13–17] have shown that local data information can still be breached through the model gradients uploaded from each client. Under the assumption that the server is *honest-but-curious*, differential privacy (DP) [18] and homomorphic encryption (HE) [19] have been introduced into FL as additional privacy-preserving mechanisms. DP injects Gaussian [20] or Laplacian [13] noise into each uploaded model gradients [21, 22], which is cost-effective and light-weighted. But the added noise has a negative impact on the model performance and cannot deal with data reconstruction attacks upon the model gradients [15].

Similar to FL, privacy preservation in distributed optimization [23, 24] is also an intrinsic issue. Mao *et al.* [25] propose a privacy preserving distributed optimization algorithm over time-varying directed communication networks by adding conditional noise to the exchanged states. Besides, Lu and Zhu [26] adopt HE to privately execute a distributed projected gradient-based algorithm on a set of agents. Different from the above work, Xu *et al.* [27] construct a federated data-driven evolutionary optimization framework to perform data-driven optimization without centrally storing the training data on the server.

Similar ideas that use HE in FL have also been presented in [28, 29], which, however, incur a considerable increase in communication costs. More recently, Zhang *et al.* [30] proposed a batchcrypt scheme for batch gradients encoding and encryption without increasing the communication costs, but this method consumes huge amount of local computational resources.

Using HE and DP [31–33] together in FL has become a popular research topic nowadays, which can defend against attacks upon the shared global model on the server side. Nevertheless, DP protection in this framework has limited effect against inference attacks from the client side, since local data has been already “masked" by

HE.

Besides HE and DP based protection methods, other techniques like secure masking has also been used. Bonawitz *et al.* [34] proposed a secure aggregation protocol by double masking uploaded local models with random numbers. In this scheme, keys are generated to do Diffle-Hellman key exchange without the help of a TTP. However, this method applies the Diffle-Hellman key exchange and Shamir secret sharing for every client, which requires a large amount of communication resources. Similarly, Kursawe *et al.* [35] proposed a single masking method for secure aggregation in smart grids without a TTP, assuming that grid collusion and party dropout do not exist, making it inapplicable to the FL environment.

Existing HE-based FL systems have the following two major drawbacks. First, most methods require a trusted third party (TTP) to generate and distribute key pairs, increasing topological complexity and attack surfaces of FL systems. In case the TTP is compromised, the secret messages will be revealed. Second, existing HE-based FL designs do not scale well to deep learning models containing a large number of parameters, because encryption and decryption of all trainable parameters are computationally prohibitive and uploading the entire encrypted model consumes a huge amount of communication resources.

In addition, privacy protection techniques used in distributed optimization are usually unsuited for federated deep learning. The main reason is that the number of model parameters in distributed optimization is much less than that in federated deep learning. Therefore, there is no high demand for secure computation in distributed optimization. Take the method used in [26] as an example, each agent (client) generates its own key pairs to encrypt its state for $n$ times without the help of TTP, where $n$ is the number of agents. The payload of $n$ times encryption is acceptable, since the state is just a scalar in distributed optimization. However, this becomes intractable in federated deep learning, as deep neural networks contain millions or even billions of model parameters.

To address the above challenges, this work aims to propose a practical and efficient privacy-preserving federated deep learning framework on the basis of additive ElGamal encryption [36] and ternary quantization of the local model parameters, DAEQ-FL for short. In DAEQ-FL, the global key pairs are collaboratively generated between the server and clients, and quantization makes it practical to encrypt deep neural networks in federated learning. The main contributions of the work are:

- We propose, for the first time, an efficient threshold encryption system with federated key generation and model quantization for federated deep learning systems. As a result, the number of model parameters to be encrypted and the communication costs for uploading the local models are considerably reduced, and an extra TTP is no longer required.

- An approximate model aggregation method is developed for the server to separately aggregate the uploaded ciphertexts and the quantized gradients, making it possible to download the aggregated ciphertexts to $T$ (threshold value) qualified clients only for distributed partial decryption. Thus, the proposed approximate aggregation can further dramatically reduce the computational and communication cost for threshold decryption.

- Extensive empirical experiments are performed to compare the proposed method to threshold Paillier [31] with respect to the learning performance, communication cost, computation time and security. Our results confirm that even encoded with 10 bits, the proposed method can significantly decrease the computation time and communication cost with no or negligible performance degradation.

The remainder of the paper is structured as follows. Section II introduces the preliminaries of federated learning, deep neural network models, and the ElGamal encryption system. Details of the proposed methods, including federated key generation, encryption and decryption, ternary quantization, and model aggregation are provided in Section III. Experimental results and discussions are given in Section IV. Finally, conclusion and future work are presented in Section V.

## 2.2. Preliminaries

### 2.2.1. Federated Learning

Unlike the centralized cloud model training that needs to collect raw data from different parties and store the data on a server, FL [3] is able to distributively learn a shared global model without accessing any private data of the clients. As shown in Fig. 2.1, at the $t$-th round of FL, $K$ connected clients download the same global model $\theta_t$ from the server and update it by training with their own data. After that, trained local models or gradients will be uploaded back to the server for model aggregation. Therefore, the global model can be learned and updated while all the training data remain on edge devices.

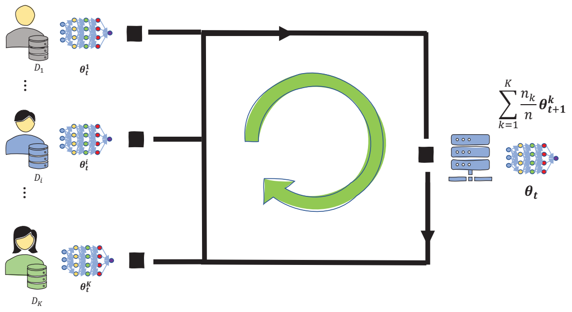


Figure 2.1: Flowchart of federated learning. $\theta_t$ are the global model parameters in the $t$-th communication round, $n_k$ is the data size of client $k$, and $K$ is the total number of clients. The global model parameters are randomly initialized at the beginning of the training and will be updated by aggregating the uploaded local models in each round.

FL aims to optimize a distributed loss function $\ell(\theta)$ as shown in Eq. (2.1),

$$\min_{\theta} \ \ell(\theta) = \sum_{k=1}^{K} \frac{n_k}{n} L_k(\theta), L_k(\theta) = \frac{1}{n_k} \sum_{i \in P_k} \ell_i(\theta, x_i) \tag{2.1}$$

where $k$ is the index of $K$ total clients, $L_k(\theta)$ is the loss function of the $k$-th local client, $n_k$ equals to the local data size, and $P_k$ is the set of data indexes whose size is $n_k$, i.e., $n_k = |P_k|$. Note that the training data $x_i$ on each client $k$ may not satisfy the independent and identically distributed assumption, i.e., non-IID.

Although the data on the clients do not need to be shared, FL is still subject to security risks, since the model gradients naturally contain information of the training data. It has been theoretically proved in [15] that only a portion of the gradients may lead to the leakage of private information of the local data. If we assume the cost function to be a quadratic function, the corresponding gradients can be calculated in Eq. (2.2).

$$\begin{aligned} &J(\theta, x) \overset{\text{def}}{=} (h_\theta(x) - y)^2 \\ &g_k = \frac{\partial J(\theta, x)}{\partial \theta} = 2(h_\theta(x) - y)\sigma^{'}(\sum_{i=1}^{d} x_i w_i + b) \cdot x_k \end{aligned} \tag{2.2}$$

where $g_k$ and $x_k$ are the $k$-th feature of gradient $g \in R^d$ and the input data $x \in R^d$, respectively. Since the product $2(h_\theta(x) - y)\sigma^{'}(\sum_{i=1}^{d} x_i w_i + b)$ is a real scalar number, the gradient is in fact proportional to the input data. As a result, uploading model gradients cannot completely prevent local data from being revealed and enhanced protection techniques are required for secure FL systems.

### 2.2.2. DEEP LEARNING

Deep learning has been deployed in the fields of computer vision, speech recognition and many other areas [37–41]. The word deep means that neural network (NN) models, such as convolutional neural networks (CNNs) [42] and recurrent neural networks (RNNs) [43], used in deep learning always contain multiple hidden layers.

For a typical supervised learning [44], the training purpose is to minimize the expected distance between a desired signal $y$ (e.g., a label in classification) and a predicted value $\hat{y}$, which is often represented by a so-called loss function $\ell(y, \hat{y})$ as shown in Eq. (2.3), where $\theta$ is the trainable model parameters we want to optimize.

$$\min_{\theta} \ \ell(\theta) = \frac{1}{N} \sum_{i} \ell(y, \hat{y} | \theta, x_i) \quad x_i \in \{x_1, x_2 ..., x_N\} \tag{2.3}$$

The stochastic gradient descent (SGD) algorithm is the most widely used optimization method that calculates the partial derivatives of the loss function (2.3) with respect to each model parameter in $\theta$. The model parameters will be updated by subtracting scaled calculated gradients as shown in Eq. (2.4),

$$\begin{aligned} g_t &= \nabla_\theta \ell(\theta, x) \\ \theta_{t+1} &= \theta_t - \eta g_t \end{aligned} \tag{2.4}$$

where $\eta$ is the learning rate and $g_t$ is the expected gradient over data samples $x$ at the $t$-th iteration. The model update based on SGD in (2.4) is repeatedly performed until the model parameters converge.

DNNs often contain a large number of layers and training very deep models on big datasets is extremely time-consuming. Therefore, DNNs will incur excessive communication and computation costs when they are adopted in FL.

### 2.2.3. Homomorphic Encryption and Secret Sharing

HE is the most widely used data protection technology in secure machine learning as it supports algebraic operations including addition and multiplication on ciphertexts. An encryption method is called partially HE if it supports addition or multiplication operation, and fully HE if it supports an infinite number of addition and multiplication operations. Without loss of security and correctness, additive HE fulfills that multiple parties encrypt message $C_i = Enc_{pk}(m_i)$ and decrypt $\sum_{i=1}^{n} m_i = Dec_{sk}(\prod_{i=1}^{n} C_i)$ using public key and secret key, respectively.

Among those well-studied HE techniques, ElGamal[36] is a multiplicative mechanism while Paillier [45] provides additive operations, which are based on discrete logarithm and composite degree residuosity classes, respectively. The former needs 256-bit key length to achieve the 128 bit security level, whereas the latter costs 3072 bits [46], implying that ElGamal is a computationally more efficient encryption and decryption method [47]. However, additional Cramer transformation [48] needs to be applied to ElGamal encryption so as to extend it to support additive operations.

Conventional HE is not well suited for distributed learning systems such as FL systems, since the ciphertexts on the server can be easily inferred, as long as one client uploads its private key to the server. In order to mitigate this issue, Adi Shamir [49] proposed Shamir Secret Sharing (SSS) which splits a secret into $n$ different shares. Consequently, $T$-out-of-$n$ shares are needed to recover the secret. Based on the SSS and DiffieHellman (DH) security definition, Feldman proposed verifiable secret sharing (VSS) [50], which adds a verification process during sending shares. The followings are the security definition.

**Definition 1** (Discrete Logarithm Hard Problem - DLHP) Discrete Logarithm is considered to be hard if

$$Pr[DLog_{\mathcal{G},\mathcal{A}}(\alpha) = 1] \leq negl(\alpha)$$

**Definition 2** (Computational Diffle-Hellman - CDH) CDH is considered to be hard if

$$Pr[\mathcal{A}(g, g^a, g^b) = (g^{ab})] \leq negl(\alpha)$$

**Definition 3** (Decisional Diffie-Hellman - DDH) DDH is considered to be hard if

$$|Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1] - Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1]| \\ \leq negl(\alpha)$$

**Definition 4** Indistinguishability - Chosen plaintext attacks (IND - CPA)

Consider a following game between an adversary $\mathscr{A}$ and a challenger:

**Set up:** Challenger generates public parameters $< g, p, q >$, public key $h$ and secret key $s$, then sends public parameters and $pk$ to the $\mathscr{A}$.

**Challenge:** $\mathscr{A}$ chooses two plaintexts $m_0$, $m_1$ with same length, then sends them to the challenger. Challenger randomly selects $b \in \{0,1\}$, encrypts $m_b$

$$C = Enc(pk, m_b)$$

and sends challenge ciphertext $C$ to $\mathscr{A}$. Finally, $\mathscr{A}$ would compute $b^{'}$.

A scheme is security against CPA if the advantage

$$\mathbf{Adv}_{\mathscr{A}}^{CPA}(\alpha) = \left| Pr(b = b^{'}) - \frac{1}{2} \right|$$

is negligible.

- Parameters generation: server runs a polynomial-time $\mathscr{G}(1^{\alpha})$ to generate public parameters $< \mathbb{G}, g, p, q >$, where $g$ is a generator of $\mathbb{G}$ which is a cyclic group with prime order $q$, $p$ is a large prime number satisfying $q|p-1$, $y$ is a random element in $\mathbb{G}$ and $\alpha$ corresponding to security level is bit length of $q$. Given a polynomial-time algorithm $\mathscr{A}$ and a negligible function $negl$, the security of key generation and encryption are based on following definitions.
- Key generation: $x \xleftarrow{\$} \mathbb{Z}_q^*$, $h = g^x \pmod{p}$.
- Encryption: $g^m \in \mathbb{Z}_p^*$, $r \xleftarrow{\$} \mathbb{Z}_q^*$, $< c_1 = g^r \pmod{p}, c_2 = g^m \cdot h^r \pmod{p} >$.
- Decryption: $\frac{c_2}{c_1^x} = \frac{mh^r}{(g^r)^x} = \frac{g^m \cdot g^{xr}}{g^{rx}} = g^m \pmod{p}$

Additive ElGamal is CPA-secure against chosen ciphertext attacks if the advantage of $\mathscr{A}$ is negligible in $\alpha$ satisfying as follows:

$$\mathbf{Adv}_{\mathscr{A}}^{CPA}(\alpha) = \left| Pr(b = b^{'}) - \frac{1}{2} \right|$$

*Proof.* Ciphertexts under additive ElGamal:

$$C = (c_1, c_2) = (g^r, g^{m_b} \cdot h^r) = g^{m_b + x \cdot r}$$

$\mathscr{A}$ computes:

$$C^{'} = C \cdot g^{m_{b'}^{-1}} = g^{x \cdot r} \; iff \; b = b^{'}$$

Besides, $\mathscr{A}$ wants to determine whether $C^{'}$ is DH agreement. $\mathscr{A}$ could "win" this game with a probability:

$$\begin{aligned} \mathbf{Adv}_{\mathscr{A}}^{ElG}(\alpha) = ||Pr[&\mathscr{A}(g, g^x, g^r, g^{x \cdot r + m_b - m_{b'}}) = 1] \\ &- Pr[\mathscr{A}(g, g^x, g^r, g^{x \cdot r}) = 1]| - \frac{1}{2}| \\ &\leq negl(\alpha) \end{aligned}$$

**Theorem 1** The exponential ElGamal and distribute key generation are CPA-secure and correctness-satisfying according to [48] and [51]. □

But Feldman VSS only allows trusted clients to share secrets and it will fail to generate correct key pairs if there are adversaries in the system. To address this limit, Perdersen [52] proposed a novel VSS that is able to detect and exclude adversarial clients. On the basis of the above two VSS methods, Gennaro *et al.* [51] introduced a secure distributed key generation (DKG) for discrete logarithm based crytosystems.

We propose a federated key generation (FKG) based on DKG for model parameter encryption in FL and the details of FKG will be given in Section III.

## 2.3. Our Proposed System

The motivation of the proposed privacy preserving system is to remove the requirement of a TTP commonly used in FL for generating key pairs for encryption, and the whole system should be robust to malicious clients. A key component for the proposed system is federated key generation (FKG) that allows the server and clients collaboratively generate key pairs without a TTP. To this end, additive discrete logarithm based encryption is adopted to achieve secure model aggregation. In addition, a fixed point encoding method is implemented to encode the plaintext. In order to decrease computational and communication resources, ternary gradient quantization and approximate model aggregation are further introduced. In the following, we elaborate each of our main components and present a description of the proposed overall DAEQ-FL system. Finally, a brief discussion is given to compare our DAEQ-FL with existing encryption based FL systems.

### 2.3.1. Threat Model

The potential threats considered in this work include those from the server, clients and outsiders in the FL proposed DAEQ-FL system. Specifically, in key generation period, we assume that at least $T$-out-of-$n$ clients generate key pairs honestly. The remaining clients could be malicious and might try to steal keys from the honest parties or make the generated keys incorrect for encryption and decryption. In the training period, we consider the server and clients are honest-but-curious, which means they strictly follow the FL algorithm but try to infer extra information from received information. Our main goal is to prevent clients data from being leaked, but the situation in which malicious clients attempt to deteriorate global model performance [53, 54] is not considered here. Besides, we assume that outsiders are passive attackers who could eavesdrop communication channels during the whole FL process.

### 2.3.2. Federated Key Generation

The proposed FKG is a variant of DKG [51], which is based on Pedersen VSS and Feldman VSS that can verify key shares in a secure way for successful key generations. The main steps of FKG are described in **Algorithm 1**.

Assume that the server in DAEQ-FL is *honest-but-curious*, and there are at least $T$-out-of-$n$ ($T > n/2$) honest clients. Before key pair generation, the server needs to generate and distribute four public parameters $p$, $q$, $g$ and $y$, where $q$ is the

**Algorithm 1:** Federated Key Generation. $\mathbb{G}$ is a cyclic group, $p$ and $q$ are large prime numbers, $g$ is a generator, $y \in \mathbb{G}$, $N$ is the number of total clients, $C$ is the fraction of clients participating the current round, $i$ is the client index, and $T$ is the threshold value.

```
Server distributes public parameters $< p, q, g, y >$
for Each FL round $t = 1, 2, \ldots$ do
    $n = C * N$
    Select threshold value $T > n/2$
    Client $i \in \{1, \cdots, n\}$ perform Pedersen VSS
    Collect the number of complaints $cpt_i$ for client $i$
    for Client $i \in \{1, \cdots, n\}$ do
        if $cpt_i > T$ then
            Mark client $i$ as disqualified
        else
            Client $i$ uploads $f_i(j)$ and
            if Eq. (2.6) is satisfied then
                Mark client $i$ as qualified (QUAL)
            else
                Mark client $i$ as disqualified
            end
            Mark client $i$ as QUAL
        end
    end
    Client $i \in$ QUAL perform Feldman VSS
    Collect complained client index in $O$,
    for Each client $i \in O$ do
        Set $counter = 0$
        for Each client $j \in$ QUAL but $j \notin m$ do
            Client $j$ uploads $f_j(i)$ and $f_j^{'}(i)$
            if Eq. (2.7) is satisfied then
                $counter = counter + 1$
            end
            if $counter \geq T$ then
                Break
            end
        end
        Retrieve $f_i(z)$ and $A_{i0}$
    end
    Generate global public key $h = \prod_{i \in QUAL} A_{i0} = g^x$
end
```

prime order of cyclic group $\mathbb{G}$, $p$ is a large prime number satisfying $p - 1 = rq$, $r$ is a positive integer, $g$ and $y$ are two different random elements in $\mathbb{G}$.

For Pedersen VSS executed in line 5 of **Algorithm** 1, each participating client $i$ in the $t$-th round generates two random polynomials $f_i(z)$ and $f_i^{'}(z)$ over $\mathbb{Z}_q^*$ of order $T-1$ as shown in Eq. (2.5).

$$\begin{aligned} f_i(z) &= a_{i0} + a_{i1}z + ... + a_{iT-1}z^{T-1} \pmod q \\ f_i^{'}(z) &= b_{i0} + b_{i1}z + ... + b_{iT-1}z^{T-1} \pmod q \end{aligned} \tag{2.5}$$

Let $z_i = a_{i0} = f_i(0)$ be the locally stored private key. Client $i$ broadcasts $C_{ik} = g^{a_{ik}}y^{b_{ik}} \pmod p$ and sends shares $s_{ij} = f_i(j)$, $s_{ij}^{'} = f_i^{'}(j)$ to client $j$ $(j \in n)$, then client $j$ verifies the received shares by *Pedersen commitment*:

$$g^{s_{ij}}y^{s_{ij}^{'}} = \prod_{k=0}^{T-1}(C_{ik})^{j^k} \pmod p \tag{2.6}$$

Due to the hiding and binding properties of Pedersen commitment [52], it is impossible for adversaries, if any, to guess the real $a_{ik}$ and $b_{ik}$ through $C_ik$ or to find another a pair of $s_{ij}$ and $s_{ij}^{'}$ that can satisfy Eq. (2.6). In addition, based on our previous security assumption and the principle of SSS, it is infeasible to reconstruct the private keys of any honest clients even if the system contains $n-T$ malicious clients.

Each client sends a complaint of client $i$ to the server if any shares $s_{ij}$ and $s_{ij}^{'}$ received from client $i$ do not satisfy Eq. (2.6). Once the server receives more than $T$ complaints against client $i$ (line 6 in **Algorithm** 1), this client will be immediately disqualified. Besides, as long as client $i$ is complained by any client $j$, where $j \in \{1, \cdots, n\}$, the corresponding shares $s_{ij}$ and $s_{ij}^{'}$ are required to upload to the central server for Pedersen commitment (Eq. (2.6)) verification. If any verification fails, client $i$ would be marked as disqualified.

However, Pedersen VSS cannot guarantee correct global public key generation, since malicious clients can still corrupt the generation process by broadcasting fake $A_{i0}$ (line 33 in **Algorithm** 1). Therefore, Feldman VSS is used in addition to Pedersen VSS to ensure that all the QUAL clients broadcast correct $A_{i0}$ for in the proposed FKG.

Similarly, to implement Feldman VSS (line 20 in **Algorithm** 1), each client $j$ ($j \in$ QUAL) broadcasts $A_{ik} = g^{a_{ik}} \pmod p$ and verifies Eq. (2.7).

$$g^{s_{ij}} = \prod_{k=0}^{T-1}(A_{ik})^{j^k} \pmod p \tag{2.7}$$

If shares of client $i$ satisfy Eq. (2.6) but not Eq. (2.7), client $j$ will send a complaint to server. Then the server requires $T$ QUAL clients to upload their shares $f_i(j), j \in t$ to retrieve the random polynomial $f_i(z)$ of client $i$ by Lagrange interpolation function [55] as show in Eq. (2.8).

$$\begin{aligned} \lambda_j &= \prod_{k \neq j} \frac{z-k}{j-k}, k \in T, j \in \text{QUAL} \\ f_i(z) &= \sum_{j \in \text{QUAL}} \lambda_j f_i(j) \end{aligned} \tag{2.8}$$

Finally, the server can generate the global public key through broadcasting $A_{i0}$ from all QUAL clients in Eq. (2.9), where $x$ is in fact the global private key. And then, the public key $h$ will be shared to all QUAL clients.

$$\begin{aligned} h_i &= A_{i0} = g^{z_i} \pmod p \\ h &= \prod_{i \in \text{QUAL}} h_i = g^{\sum_{i\in\text{QUAL}} z_i} = g^x \pmod p \end{aligned} \tag{2.9}$$

Pedersen's VSS and Feldman's VSS for FKG require at most $768NT + 64N(N-1)$ bytes of communication, where $N$ is the total number of clients and $T$ is the threshold.

### 2.3.3. ADDITIVE DISCRETE LOGARITHM BASED ENCRYPTION

To be fully compatible with FKG, which is adapted from DKG to the FL environment, additive discrete logarithm based encryption is employed based on ElGamal encryption [36].

The original ElGamal encryption works as follows:

- Parameters generation: Generate three parameters $p$, $q$ and $g$, where $q$ is the prime order of a cyclic group $\mathbb{G}$, $p$ is a large prime number satisfying $q|p-1$, and $g$ is a generator of $\mathbb{G}$.
- Key generation: Select a random number $x, x \in \mathbb{Z}_q^*$ as the secret key, and then compute $h = g^x \pmod p$ to be the public key.
- Encryption: To encrypt a message $m \in \mathbb{Z}_p^*$, choose a random number $r \in \mathbb{Z}_q^*$ as a ephemeral key, calculate two ciphertexts as $< c_1 = g^r \pmod p, c_2 = mh^r \pmod p >$.
- Decryption: The plaintext message $m$ can only be decrypted if the private key $x$ is available by computing Eq. (2.10)

$$\frac{c_2}{c_1^x} = \frac{mh^r}{(g^r)^x} = \frac{mg^{xr}}{g^{rx}} \pmod p \equiv m \tag{2.10}$$

Therefore, the original ElGamal is a multiplicative HE satisfying: $\text{Enc}(m1) * \text{Enc}(m2) = \text{Enc}(m1 * m2)$, because $m_1 h^{r_1} * m_2 h^{r_2} = m_1 m_2 h^{r_1+r_2} \pmod p$. Since model aggregation on the server in FL performs the addition operation, we can apply Cramer transformation [48] on ElGamal encryption by simply converting the plaintext $m$ into $m^{'} = g^m \pmod p$. Consequently, the original ElGamal encryption becomes a discrete logarithm based additive HE, as shown in Eq. (2.11).

$$\begin{aligned} \text{Enc}(m_1) * \text{Enc}(m_2) &= g^{m_1} h^{r_1} * g^{m_2} h^{r_2} \\ &= g^{m_1+m_2} h^{r_1+r_2} \pmod p \end{aligned} \tag{2.11}$$

A security analysis is described in the following.

- Parameters generation: server runs a polynomial-time $\mathcal{G}(1^{\alpha})$ to generate public parameters $<\mathbb{G}, g, p, q>$, where $g$ is a generator of $\mathbb{G}$ which is a cyclic group with prime order $q$, $p$ is a large prime number satisfying $q|p-1$, $y$ is a random element in $\mathbb{G}$ and $\alpha$ corresponding to security level is bit length of $q$. Given a polynomial-time algorithm $\mathscr{A}$ and a negligible function $negl$, the security of key generation and encryption are based on following definitions.

- Key generation: $x \xleftarrow{\$} \mathbb{Z}_q^*$, $h = g^x \pmod p$.

- Encryption: $g^m \in \mathbb{Z}_p^*$, $r \xleftarrow{\$} \mathbb{Z}_q^*$, $< c_1 = g^r \pmod p, c_2 = g^m \cdot h^r \pmod p >$.

- Decryption: $\frac{c_2}{c_1^x} = \frac{mh^r}{(g^r)^x} = \frac{g^m \cdot g^{xr}}{g^{rx}} = g^m \pmod p$

Additive ElGamal is CPA-secure against chosen ciphertext attacks if the advantage of $\mathscr{A}$ is negligible in $\alpha$ satisfying as follows:

$$\mathbf{Adv}_{\mathscr{A}}^{CPA}(\alpha) = \left| Pr(b = b') - \frac{1}{2} \right|$$

*Proof.* Ciphertexts under additive ElGamal:

$$C = (c_1, c_2) = (g^r, g^{m_b} \cdot h^r) = g^{m_b + x \cdot r}$$

$\mathscr{A}$ computes:

$$C' = C \cdot g^{m_{b'}^{-1}} = g^{x \cdot r} \ iff \ b = b'$$

Besides, $\mathscr{A}$ wants to determine whether $C'$ is DH agreement. $\mathscr{A}$ could "win" this game with a probability:

$$\begin{aligned}\mathbf{Adv}_{\mathscr{A}}^{ElG}(\alpha) = ||Pr[\mathscr{A}(g, g^x, g^r, g^{x \cdot r + m_b - m_{b'}}) = 1] \\ - Pr[\mathscr{A}(g, g^x, g^r, g^{x \cdot r}) = 1]| - \frac{1}{2}| \\ \leq negl(\alpha)\end{aligned}$$

**Theorem 1** The exponential ElGamal and distribute key generation are CPA-secure and correctness-satisfying according to [48] and [51]. □

### 2.3.4. Fixed Point Encoding Method

Note that HE can be applied to integers only, however, model parameters or gradients are normally real numbers. Therefore, the real-values model parameters must be encoded before encryption.

The encoding method used in this work is straightforward, as shown in Eq. (2.12), where $s_t$ is the maximum absolute of the gradients (will be introduced in Section

2.3.6), $b$ is the encoding bit length, $q$ is the above mentioned prime order of $\mathbb{G}$ and $m$ is the encoded integer number.

$$\begin{aligned}
&\hat{m} = \text{round}(s_t * 2^b), |\hat{m}| \le int_{\max} \\
&\text{Encode}(s_t) = \hat{m} \pmod q = m \\
&\text{Decode}(m) = \begin{cases} m * 2^{-b}, & m \le int_{\max} \\ (m-q) * 2^{-b}, & m > q - int_{\max} \end{cases}
\end{aligned} \tag{2.12}$$

$int_{\max}$ is the maximum positive encoding number defined by the server, which is one-third of $q$. If $m > q - int_{\max}$ ($\hat{m}$ is negative), it should subtract $q$ before multiplying $2^{-b}$, because $\hat{m} \pmod q = q + \hat{m}, \hat{m} < 0$. Since the bit length of $int_{\max}$ is always set to be much larger than the encoding bit $b$, sufficient value space can be reserved for encoding number summations (additive HE).

## 2.3.5. Brute Force and Log Recovery

Using Cramer transformation needs to recover the desired $m$ from $m'$ to solve the so-called discrete logarithm hard problem after decryption. Here, we propose two techniques, namely brute force and log recovery, to solve this problem.

Brute force recovery simply tries different $m$ from 0 to $q-1$, and the correct $m$ is found only if $m' = g^m \pmod p$. Thus, a maximum of $q$ trials are needed to solve DLHP in the worst case. Fortunately, the absolute values of all the model gradients in DNNs are always less than 0.1, and therefore, they can be encoded with a $b$ bit-length fixed point integer number. It takes about $2^b$ times to find the correct $m$ if $\hat{m}$ is positive. Note that the quantization method we employ can guarantee that $\hat{m}$ is positive, which will be introduced later.

The log method consumes almost no additional recovery time by calculating $\log_g(g^m) = m$ directly. However, this only works when $g^m < p$. In order to ensure the best encoding precision (the encoded $m$ should be as large as possible), we minimize $g$ to $g_0 = 2$. And the encoded number $m$ must be less than the bit length of a large prime number $p$, which means the encoding precision is restricted by the security level. Besides, the security level will not be reduced by selecting a small fixed $g_0$.

Both the brute force and log recovery are described in **Algorithm 2**

## 2.3.6. Ternary Gradient Quantization

Encrypting and decrypting all elements of the model gradients have several shortcomings. First, performing encryption on local clients is computationally extremely expensive, causing a big barrier for real world applications, since usually the distributed edge devices do not have abundant computational resources. Second, uploading model gradients in terms of ciphertext incurs a large amount of communication costs. Finally, the first two issues will become computationally prohibitive when the model is large and complex, e.g., DNNs.

In order to tackle the above challenges, we introduce ternary gradient quantization (TernGrad) [56] to drastically reduce computational and communication costs for

**Algorithm 2:** Plaintext Recovery. $q$ is a prime order of the cyclic group $\mathbb{G}$, $p$ is a large prime number satisfying $p-1|q$, $g$ is a random element in $\mathbb{G}$, $m$ is the message to be recovered, $int_{\max}$ is the maximum positive encoded number.

```
1  **Brute Force Recovery:**
2  $g_0 = g$
3  Given decrypted plaintext $m' = g_0^m \pmod p$, $m \le int_{\max}$
4  for $j$ from 0 to $q-1$ do
5      if $g_0^j \pmod p == m'$ then
6          $m = j$ Break
7      else
8          Continue
9      end
10 end
11 Return $m$
12 **Log Recovery:**
13 $g_0 = 2$
14 Given decrypted plaintext $m' = g_0^m \pmod p$, $g_0^m \le p$
15 $m = \log_{g_0}(g_0^m)$
16 Return $m$
```

encryption of DNNs. TernGrad compresses the original model gradients into ternary precision gradients with values $\in \{-1,0,1\}$ as described in Eq. (2.13),

$$\begin{aligned}\tilde{g}_t &= s_t \cdot \text{sign}(g_t) \cdot b_t \\ s_t &= \max(\text{abs}(g_t))\end{aligned} \tag{2.13}$$

where $g_t$ is the full precision model gradients at the $t$-th iteration, $\tilde{g}_t$ is the quantized gradients, $s_t$ (a scalar larger than 0) is the maximum absolute element among $g_t$, and the $sign$ function transfers $g_t$ into binary precision with values $\in \{-1,1\}$. Finally, $b_t$ is a binary tensor whose elements follow the Bernoulli distribution [57].

$$\begin{aligned}Pr(b_{tk} = 1|g_t) &= |g_{tk}|/s_t \\ Pr(b_{tk} = 0|g_t) &= 1 - |g_{tk}|/s_t\end{aligned} \tag{2.14}$$

where $b_{tk}$ and $g_{tk}$ is the $k$-th element of $b_t$ and $g_t$, respectively, and the product of sign($g_t$) and $b_{tk}$ is a ternary tensor ($\{-1,0,1\}$) representing the model training direction. According to Eq. (2.4), the full precision model parameters in TernGrad is updated as shown in Eq. (2.15).

$$\theta_{t+1} = \theta_t - \eta(s_t \cdot \text{sign}(g_t) \cdot b_t) \tag{2.15}$$

Since $s_t$ is a random variable depending on input $x_t$ and the model weights $\theta_t$, Eq. (2.14) can be re-written into Eq. (2.16).

$$\begin{aligned}Pr(b_{tk} = 1|x_t, \theta_t) &= |g_{tk}|/s_t \\ Pr(b_{tk} = 0|x_t, \theta_t) &= 1 - |g_{tk}|/s_t\end{aligned} \tag{2.16}$$

The unbiasedness of the ternary gradients can be proved as shown in Eq. (2.17).

$$\begin{aligned}\mathbb{E}(s_t \cdot \mathrm{sign}(g_t) \cdot b_t) &= \mathbb{E}(s_t \cdot \mathrm{sign}(g_t) \cdot \mathbb{E}(b_t|x_t)) \\ &= \mathbb{E}(g_t)\end{aligned} \tag{2.17}$$

The TernGrad algorithm is adopted in the proposed DAEQ-FL to significantly reduce the computational cost for encryption on local devices and the communication costs for passing the encrypted model parameters between the clients and the server. Before performing encryption on the local clients, the model gradients are decomposed into two parts: one is the positive scalar $s_t$ and the other is the ternary model gradients $\mathrm{sign}(g_t) \cdot b_t$). And only one scalar $s_t$ in each layer of the model needs to be encrypted, thereby dramatically decreasing the computation costs and encryption time.

The ternary gradients do not need to be encrypted, because adversaries, if any, can only derive parts of gradients sign information from them. It should be noted that existing popular attacks [15, 58–63] on FL do not have ability to retrieve the original data from ternary gradients. In addition, separately uploading encrypted $s_t$ and ternary gradients reduce the total amount of uploads to 16 times smaller than the original. Another advantage is that it can make brute force recovery much faster, since the scalar $s_t$ in Eq. (2.12) can never be negative.

### 2.3.7. Approximate Model Aggregation

For model aggregation on the server, the received encrypted $s_t$ ($t$-th round in FL) should multiply its corresponding ternary gradients before weighted averaging (Eq. (2.18)).

$$\begin{aligned} g^i_{(t,\mathrm{tern})} &= \mathrm{sign}(g^i_t) \cdot b^i_t \\ \mathrm{Enc}(g^{\mathrm{global}}_t) &= \sum_i \frac{n_i}{n} (\mathrm{Enc}(s^i_t) * g^i_{(t,\mathrm{tern})}) \end{aligned} \tag{2.18}$$

Decryption of the aggregated global gradients $\mathrm{Enc}(g^{\mathrm{global}}_t)$ requires to traverse each single ciphertext, which is extremely time-consuming, making it less practical even if the server often possess abundant computation resources. Therefore, this work proposes an approximate aggregation method (Fig. 2.2) so as to reduce the decryption time. The basic idea is to separately aggregate the encrypted scalar and related ternary gradients, as shown in Eq. (2.19).

$$\begin{aligned} \mathrm{Enc}(s^{\mathrm{global}}_t) &= \prod_i \mathrm{Enc}(s^i_t * \frac{n_i}{n}) = \mathrm{Enc}(\sum_i s^i_t * \frac{n_i}{n}) \\ g^{\mathrm{global}}_{(t,\mathrm{tern})} &= \sum_i g^i_{(t,\mathrm{tern})} \end{aligned} \tag{2.19}$$

where $i$ is the client index, $n_i$ is the local data size and $n$ is the global data size. And $\mathrm{Enc}(g^{\mathrm{global}}_t) = \mathrm{Enc}(s^{\mathrm{global}}_t) * g^{\mathrm{global}}_{(t,\mathrm{tern})}$, if $s^1_t = s^2_t = ... = s^n_t$. However, this condition is hard to satisfy and the bias $g^{\mathrm{global}}_t - s^{\mathrm{global}}_t * g^{\mathrm{global}}_{(t,\mathrm{tern})}$ is difficult to estimate due to the

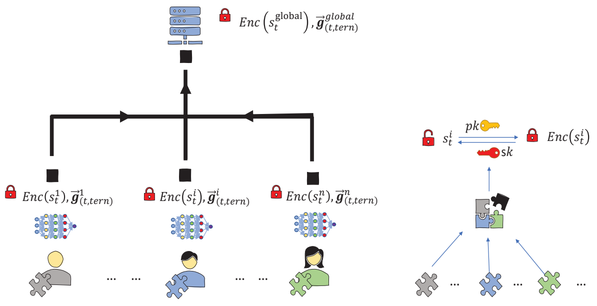

$Enc\left(s_t^{\text{global}}\right), \vec{g}_{(t,tern)}^{global}$
$Enc(s_t^1), \vec{g}_{(t,tern)}^1$
$Enc(s_t^i), \vec{g}_{(t,tern)}^i$
$Enc(s_t^n), \vec{g}_{(t,tern)}^n$
$s_t^i$
$pk$
$sk$
$Enc(s_t^i)$

Figure 2.2: Encryption with TernGrad and model aggregation approximation. The $\text{Enc}(s_t^{\text{global}})$ is aggregated over the uploaded $\text{Enc}(s_t^i)$ from the participating clients.

random property of SGD. In reality, each client's local scalar satisfies $s_t^1 \approx s_t^2 \approx ... \approx s_t^n$, and our experimental results (Section II.B in the Supplementary material) confirm that the values of $s_t$ of different clients are very similar, making this approximation reasonable.

Note that a small implementation trick used here is to do weighted averaging upon $s_t$ before encryption, which helps reduce the differences between $s_t^i$ and avoid overflow of the encrypted gradients during model aggregations on the server.

To update the global model, only two ciphertexts $\text{Enc}(s_t^{\text{global}})$ of each layer need to be decrypted, which will be multiplied by the global ternary gradients afterwards as shown in Eq. (2.20)

$$
\begin{aligned}
s_t^{\text{global}} &= \text{Dec}(\text{Enc}(s_t^{\text{global}})) \\
\theta_{t+1}^{\text{global}} &= \theta_t^{\text{global}} - \eta s_t^{\text{global}} * g_{(t,\text{tern})}^{\text{global}}
\end{aligned}
\tag{2.20}
$$

## 2.3.8. OVERALL FRAMEWORK: DAEQ-FL

The overall framework, distributed additive ElGamal encryption and quantization for privacy-preserving federated deep learning, DAEQ-FL for short, is depicted in **Algorithm** 3. Note that our algorithm generates key pairs at the beginning of each round, considering that in practice the keys are often frequently changed. This is, however, not a mandatory requirement.

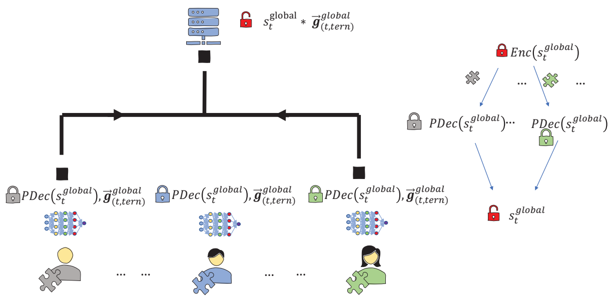
Note that the server can determine the threshold value $T$ based on the number of participating clients $n$ in each FL round ($T > n/2$). If the number of QUAL clients are less than $T$, the disqualified clients will be kicked out of the system and then the process is aborted and FKG is restarted. After FKG, each QUAL client $i$ downloads the global model parameters $\theta_t$ and the public key $pk$ for local training. Then the model gradients, obtained by subtracting the received global model $\theta_t$ from the local updated model $\theta_t^i$, are converted into a real-valued coefficient $s_t^i$ and a ternary matrix $\Delta\theta_{(t,\text{tern})}^i$ before performing ElGamal encryption. Two ciphertexts $c_{(t,1)}^i$ and $c_{(t,2)}^i$ together with a ternary gradient $\Delta\theta_{(t,\text{tern})}^i$ are then uploaded to the server for

**Algorithm 3:** DAEQ-FL. $p$, $q$, $g$, $y$ are key parameters introduced in **Algorithm 1**, $pk$ is the global public key, $Qual$ are qualified clients, $N$ is the total number of clients, $C$ is the fraction of connected clients, $E$ is the number of local epochs, $B$ is the local batch data, $\theta_t$ is the global model parameters at the $t$-th FL round, $T$ is the threshold value, and $\eta$ is the learning rate.

```
Server:
Generate and distribute $p$, $q$, $g$, $y$ and global model parameters $\theta_0$
for each FL round $t = 1, 2, \dots$ do
    Select $n = C \times N$ clients, $C \in (0,1)$
    Select $T > n/2$
    Generate $pk$ by FKG among $n$ clients in Algorithm 1
    for each client $i \in Qual$ in parallel do
        Download $\theta_t$
        Do local Training
        Upload $c^i_{(t,1)}$, $c^i_{(t,2)}$ and $\Delta\theta^i_{(t,tern)}$
    end
    $c_{(t,1)} = \prod_i c^i_{(t,1)} \pmod p$
    $c_{(t,2)} = \prod_i c^i_{(t,2)} \pmod p$
    $\Delta\theta_{(t,tern)} = \sum_i \Delta\theta^i_{(t,tern)}$
    Randomly select $T$ $Qual$ clients
    for each client $j \in T$ in parallel do
        Download $c_{(t,1)}$ and $c_{(t,2)}$
        Do Partial Decryption
    end
    $g_0^{Tm_t} = \prod_{j \in T} pd_j = g_0^{Tm_t} g^{(x - \sum_j \lambda_j x_i)T} \pmod p$
    Recover $Tm_t$ by Algorithm 2
    $\theta_{t+1} = \theta_t - \Delta\theta_{(t,\mathrm{tern})} * Tm_t / T$
end
Client $i$:
// Training:
$\theta^i_t = \theta_t$
for each iteration from 1 to $E$ do
    for batch $b \in B$ do
        $\theta^i_t = \theta^i_t - \eta \nabla L_i(\theta^i_t, b)$
    end
end
$\Delta\theta^i_t = \theta^i_t - \theta_t$
Quantize $\Delta\theta^i_t$ into $s^i_t$ and $\Delta\theta^i_{(t,\mathrm{tern})}$ in Eq. (2.13) and (2.18)
Encode $m^i_t = \mathrm{round}(s^i_t * D_k / D * 2^l) \pmod q$ in Eq. (2.12)
Encrypt $c^i_{(t,1)} = g^{r_i} \pmod p$, $c^i_{(t,2)} = g_0^{m^i_t} pk^{r_i} \pmod p$
Return $c^i_{(t,1)}$, $c^i_{(t,2)}$ and $\Delta\theta^i_{(t,tern)}$ to server
// Partial Decryption:
$x_i = \sum_j s_{ji} = f_j(i)$, $i, j \in \mathrm{QUAL}$
Partial decrypt $pd_i = c_{(t,2)} / c_{(t,1)}^{\lambda_i x_i T} \pmod p$
Return $pd_i$
```

model aggregation as described in Section 2.3.7.

For decryption (Fig. 2.3), only two aggregated ciphertexts need to be downloaded to $T$ QUAL clients for partial decryption and $T$ partial decrypted ciphertexts $pd_i$ are uploaded back to the server (line 16-20 in **Algorithm 3**). The server can easily get the plaintext $g_0^{Tm_t}$ by multiplying all received ciphertexts $pd_i$. $s_{ji}$ is a share $f_j(i)$



$s_t^{\text{global}} * \vec{g}_{(t,tern)}^{global}$
$Enc(s_t^{global})$
...
...
$PDec(s_t^{global})$...
$PDec(s_t^{global})$
$s_t^{global}$
$PDec(s_t^{global}), \vec{g}_{(t,tern)}^{global}$
$PDec(s_t^{global}), \vec{g}_{(t,tern)}^{global}$
$PDec(s_t^{global}), \vec{g}_{(t,tern)}^{global}$
... ...
... ...

Figure 2.3: The encrypted $s_t^{\text{global}}$ is downloaded to $T$ qualified clients for partial decryption, and then the partial decrypted ciphertexts are uploaded back to the server to retrieve the final plaintext.

(introduced in **Algorithm 1**) from client $j$ to client $i$, $\lambda_i = \prod_{j \neq i} \frac{j}{j-i}$ is the Lagrange coefficient and $x = \sum_{j \in \text{QUAL}} z_j$ is the global private key. According to the property of SSS, at least $T$ (threshold value) different $f_i(j), j \in T$ shares are needed to retrieve the local private key $z_i$. The reason why $x - \sum_i \lambda_i x_i = 0$ is proved below:

$$\begin{aligned}\sum_{i \in T} \lambda_i x_i &= \sum_{i \in T} \lambda_i \sum_{j \in \text{QUAL}} f_j(i) \\ &= \sum_{j \in \text{QUAL}} \sum_{i \in T} \lambda_i f_j(i) \\ &= \sum_{j \in \text{QUAL}} z_j = x\end{aligned} \tag{2.21}$$

One of the advantages of DAEQ-FL is that no parties within the FL system, including the server, can know the global private key $x$, which significantly enhances system security level. In addition, the extra communication resources are negligible, and only three ciphertexts $c_{(t,1)}$, $c_{(t,2)}$ and $pd_i$ are transmitted between the server and each client $i$ with the help of the TernGrad algorithm. Finally, DAEQ-FL is robust to possible disconnection of individual clients, since the ciphertext can be successfully decrypted so long as a minimum of $T$ QUAL clients upload their $pd_i$.

### 2.3.9. Discussion

We list the general differences between our proposed system and four popular existing approaches in Table 2.1. From the table we can see that our DAEQ-FL has remarkable superiority, being a threshold based encryption system without an extra TTP and tolerating the existence of malicious clients. We here note that Truex *et al.* [31] also proposed a threshold based Paillier encryption system in FL, but it still requires a TTP for key generation. Some quantization technique is introduced in [30] for efficient encoding and encryption, but it is totally different from our ternary quantization methods.

Compared to LWE [64, 65] causing excessive message expansion and functional encryption [66, 67] with large computaional complexity, Paillier is a relatively lightweighted encryption scheme. Therefore, we compare Paillier with our system in the next section. It is worth considering that in the real world scenario, outsiders could tamper the messages communicating between server and clients, a signature scheme could be used here [36] based on keys generated by FKG. In addition, the double masking method proposed by Bonawitz *et al.* [34] needs to recover masking values as long as one client is offline which is not computationally efficient. Our threshold based encryption scheme is more efficient and robust to this drop out problem, since it only require T-out-of-n clients to be online during decryption period.

Table 2.1: Comparison of encryption based privacy preserving FL systems

| Proposed systems | Threat model | | Encryption scheme | Without TTP | Threshold Based |
|---|---|---|---|---|---|
| | Server | Client | | | |
| Phong *et al.* [15] | honest but curious | honest | Paillier, LWE | ✗ | ✗ |
| Truex *et al.* [31] | honest but curious | majority honest | Paillier | ✗ | ✓ |
| Xu *et al.* [68] | honest but curious | majority honest | functional encryption | ✗ | ✗ |
| Batchcrypt[30] | honest but curious | honest | Paillier | ✓ | ✗ |
| Ma *et al.* [69] | honest but curious | honest | ElGamal | ✓ | ✗ |
| DAEQ-FL(our system) | honest but curious | majority honest | ElGamal | ✓ | ✓ |

## 2.4. Experimental Results

In this section, we first introduce the datasets and models used in the experiments, together with all settings of the FL and encryption. We also present the communication and computation cost resulting from encryption, as well as the time consumption of the brute force recovery, followed by the analysis of approximate aggregation and a description of results on the model performances. Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz and NVIDIA Quadro RTX 6000 GPU are used in our experiments.

### 2.4.1. Dataset and Model Information

In our simulations, we use CNN for MNIST [70] digit number classifications, ResNet for CIFAR10 [71] image classifications and stacked LSTM [72] for Shakespeare [73] next word prediction task. All three datasets are non-iid among different clients.

MNIST is a 28x28 grey scale digit number image dataset containing 60,000 training images and 10,000 testing images with 10 different kinds of label classes (0~9). All the clients' training data are distributed according to their label classes and most clients contain only two kinds of digits for non-iid partition.

CIFAR10 contains 10 different kinds of 50,000 training and 10,000 testing 32x32x3 images. Similar to MNIST, the whole training data are horizontally sampled and each client owns five different kinds of object images.

The Shakespeare dataset is built from the whole work of Wailliams Shakespeare. It has in total 4,226,073 samples with 1129 role players and the data samples of each role player represent the dataset on each client. Additionally, 90% of the user's data are randomly divided as the training data and the rest are testing data. This dataset is naturally non-iid and unbalanced, with some clients having few lines and others a large number of lines. In order to reduce the training time, we follow the method used in [74] to randomly select 5% of the total users and remove those containing less than 64 samples.

Note that we do not apply any data augmentation techniques [75, 76] to boost the final global model performances to reduce local computational complexity, since the main purpose of this work is not to achieve the state of the art model performances in FL; instead, we aim to present a distributed encryption method for better privacy preservation in FL without considerably increasing the computational and communication costs.

A CNN model is adopted to train on MNIST in the FL framework, which contains two 3x3 convolution layers with 32 and 64 filter channels, respectively, followed by a 2x2 max pooling layer. And then, a hidden layer with 128 neurons is fully connected to the flattened output of the max pooling layer. Thus, the whole CNN model has 1,625,866 learnable parameters.

CIFAR10 dataset is to be learned by a ResNet model. The input images firstly pass through a 3x3 convolutional layer with 64 channels, followed by a batch normalization layer [77] with the Relu [78] activation function. Its output is connected to four sequentially connected block layers with 64, 128, 256, 512 filter channels, respectively. Each block layer contains two residual blocks containing two convolutional layers, each followed by a batch normalization layer and a shortcut connection. All the trainable parameters of the batch normalization layers are disabled, because they are observed to perform poorly with small batch sizes and non-iid data [77, 79]. The full ResNet model has 11,164,362 trainable parameters.

The Shakespeare dataset is trained by a stacked LSTM model which contains two LSTM layers, each with 256 neurons. Since we use the module cudnnLSTM of Tensorflow [80] , the layer bias is twice as large as the original LSTM layer. Thus, the full model contains 819,920 parameters.

### 2.4.2. Federated Learning and Encryption Settings

In the experiments for image classification using CNN and ResNet , the FL system consists of total 20 clients, each containing 3000 and 2500 data samples for MNIST and CIFAR10, respectively. The total number of communication rounds is set to be 200 and all the clients are connected to the server in each communication round. In the experiments for language modeling using the LSTM, we randomly sample 5% of the entire role players (36 role users containing at least 64 samples) in Shakespeare dataset. Therefore, only 10 out of the 36 clients are randomly chosen to participate the training, following the settings in [74]. The total number of communication rounds is set to be 100.

We use SGD algorithm for all model training. For the CNN models, the number of local epochs is set to 2, the batch size is 50, and the learning rate is 0.1 with a decay rate of 0.995 over the FL rounds. We do not use any momentum for training the CNNs, while the momentum is set to be 0.5 for the ResNet. For the LSTM, the local epoch is set to 1, the batch size is 10, and the learning rate is 0.5 with a decay rate of 0.995.

The threshold rate is set to be 0.6 and the corresponding threshold value $T$ is $0.6n$, where $n$ is the number of connected clients in each communication round. Note that threshold $T$ can be set to any value so long as it satisfies the condition $T > n/2$ so that the assumption that the number of benign clients is larger than that of malicious clients holds. The key size and group size of the distributed additive ElGamal encryption are set to 256 and 3072, respectively, to offer a 128-bit security level. Besides, the bit length $b$ for encoding is chosen to be from 2 to 15. Due to the limited computation resources, a very large encoding bit length ($b > 15$) is not used here, since it will consume much more time for brute force recovery. Besides, for the log recovery, $g_0$ is set to 2 for fast encryption and the condition $g_0^m \leq p$ must be satisfied as described in Algorithm 2. Since the group size of $p$ is 3072, the encoded number $m$ must satisfy the condition $2^m \leq 3072$ and the corresponding encoding bit length satisfies the condition $b \leq 11.6$. Also considering the overflow problem previously discussed, the largest encoding bit length for log recovery is set to 10. As a result, the log recovery is used when the encoding bit length ranges from 2 to 10, while the brute force recovery method is adopted when the bit length is larger than 10.

### 2.4.3. Encryption Cost and Brute Force Recovery Time

At first, we compare the communication costs between the proposed DAEQ-FL and a threshold based Paillier method in terms of encryption and recovery costs.

Table 2.2: Communication costs of one connected client for both encryption and partial decryption with 128-bit security level, # of ciphertexts means the number of transmitted ciphertexts for encryption and decryption, respectively.

| Models | Enc Uploads (MB) | Dec Downloads (MB) | Dec Uploads (MB) | # of Ciphertexts |
|---|---|---|---|---|
| CNN (DAEQ-FL) | 0.3876+0.0059 | 0.0059 | 0.0029 | 16+24 |
| ResNet (DAEQ-FL) | 2.6618+0.0161 | 0.0161 | 0.0081 | 44+66 |
| LSTM (DAEQ-FL) | 0.1955+0.0066 | 0.0066 | 0.0033 | 18+27 |
| CNN (Paillier) | 595.4099 | 595.4099 | 595.4099 | 1625866+3251732 |
| ResNet (Paillier) | 4088.5115 | 4088.5115 | 4088.5115 | 11164362+22328724 |
| LSTM (Paillier) | 300.2637 | 300.2637 | 300.2637 | 819920+1639840 |

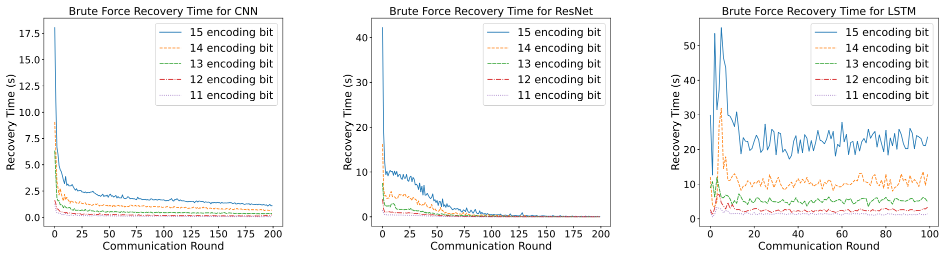
We experiment with three different models (CNN, ResNet and LSTM) in our DAEQ-FL system and compare them with the Paillier-based variants. As shown in Table 2.2, the communication cost of our system is dramatically less than those Paillier-based systems. The best case has been achieved in the LSTM, where each client consumes only 0.212MB of communication costs in one round, whereas the Paillier system takes 900.7911MB, which is about 4249x of our system. For training the CNN and ResNet, the proposed DEAQ-FL costs 0.4023MB and 2.7021MB, respectively, which accounts for approximately 0.023% and 0.022% of the Paillier based variants.

Next, we compare the runtime of the ElGamal encryption used in our system and the conventional Paillier method under the same security level. The runtimes of the two encryption methods for encrypting and decrypting one number are listed in Table 2.3, where both the key size of Paillier and the group size of ElGamal are 3072.

Table 2.3: Runtime for encryption and decryption of one number using ElGamal and Paillier

| Algorithm | Enc Time (s) | Dec Time (s) |
|---|---|---|
| ElGamal | 0.0029 | 0.0015 |
| Paillier | 0.0501 | 0.0141 |

From the results in Table 2.3, we can observe that ElGamal is approximately 17 times and 10 times faster than Paillier for encryption and decryption, respectively. However, since the Cramer transformation needs extra brute force recovery time, in the following, we explore the brute force recovery time corresponding to different encoding bit lengths for CNN, ResNet18 and stacked LSTM, respectively. The comparative results are plotted in Fig. 2.4.



(a) CNN for MNIST (b) ResNet for CIFAR10 (c) LSTM for Shakespeare

Figure 2.4: Brute force recovery time with different encoding bit lengths for different learning models.

Because the larger the encoding bit length is, the more time the brute force recovery will consume. Here, we experiment with the encoding bit length starting from 15 bits to 2 bits. The results clearly show that the difference in computation times goes bigger with rising of encoding bit. Specifically, CNN, ResNet and LSTM spend at most 18.0467s, 42.2203s and 55.2088s on recovery, respectively.

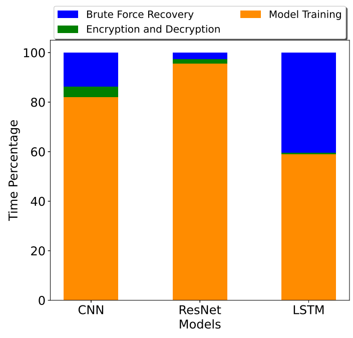
The CNN and ResNet show similar recovery time profile over the communication rounds. Their brute force recovery time are very large in the beginning, and quickly drop over the communication rounds. This is attributed to the fact that the gradients of the model parameters of the SGD decrease quickly as the global model parameters converge. It is surprisingly to see that the recovery time for the ResNet becomes almost zero, which is smaller than that of the CNN after approximately 100 communication rounds. This means the model gradients of the ResNet become very small at the end of federated model training.

By contrast, the recovery time of the LSTM does not drop to zero and keeps fluctuating at a relatively high level, especially for the 15-bit length encoding. There are two reasons for the above observations. First, training of the LSTM involves large gradients of recurrent connections, and those values are determined by the length of sequence. Second, the setting of the FL environment for the LSTM is very different from that of the CNN and ResNet, where only ten clients randomly participate global model aggregation in one communication round.

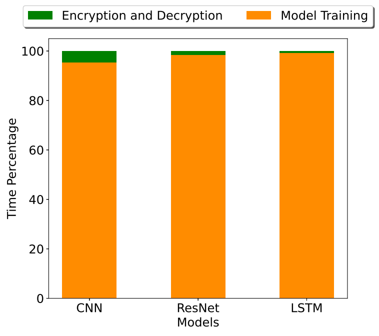
Table 2.4 presents the time consumption of the brute force recovery for 15-bit encoding length. From Fig. 2.5(a) we can find that the average brute force recovery time accounts for a great proportion of the total elapsed time in each communication round, especially for the LSTM.

Table 2.4: Brute force recovery time for 15 encoding bit length

| Models | Max (s) | Min (s) | Avg (s) |
|---|---|---|---|
| CNN | 18.0467 | 1.0717 | 1.9786 |
| ResNet | 42.2203 | 0.0474 | 2.6491 |
| LSTM | 55.2088 | 12.6166 | 23.8876 |



(a) 15 bit Brute Force Recovery



(b) 10 bit Log Recovery

Figure 2.5: Ratio of consumed time for model training, encryption and decryption, and brute force recovery for the 15-bit encoding length (a), and for the log recovery for the 10-bit encoding length (b) in one communication round.

Therefore, we can use the log recovery instead of the brute force recovery when the encoding bit length is smaller than or equal to 10 so that the recovery time and the encryption time become negligible (Fig. 2.5(b)). Since the group size of $p$ is 3072 bit and $g_0$ is 2, the log recovery is not recommended when the plaintext message is larger than 3072 bit (11-bit encoding length). In order to avoid overflow, the maximum encoding bit length is set to be 10 in our simulations and the global performance drop caused by a low encoding bit length will be discussed in the next section.

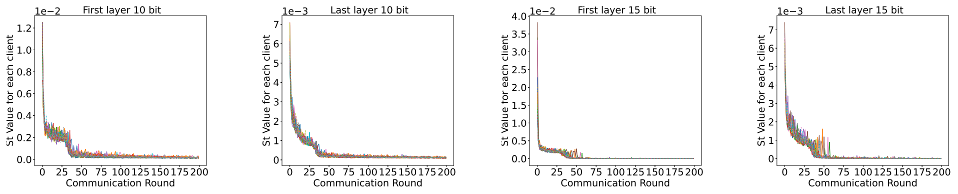
## 2.4.4. Analysis of Approximate Model Aggregation

For approximate aggregation analysis, we vary the degrees of data skew among each client, since the local gradient differences mainly come from different local data distributions. More specifically, there are three kinds of different local data distributions: 1) independent and identical distributed (iid): the training data is randomly allocated to each client, 2) non-iid with 5 classes: each client owns five different kinds of object images, 3) non-iid with 2 classes: each client owns two different kinds of images.

ResNet is adopted for this analysis, since it is the most complex model containing the largest trainable parameters in our experiments and the results derived from ResNet are more representative. In addition, we just display the maximum absolute elements $s_t$ of the first convolutional layer and the last fully connected layer, because it is redundant and unnecessary to show total 22 $s_t$ values of all 20 layers in one client. Except that, these experiments are performed under the federated encryption environment with our DAEQ-FL algorithm and two kinds of encoding bit length (10 and 15) are used here.

The purpose of this ablation study is to observe whether the values of $s_t$ of different clients are similar. And if $s_t^1 \approx s_t^2 \approx ... s_t^n$ holds, the bias between $g_t^{\text{global}}$ and $s_t^{\text{global}} * g_{(t,\text{tern})}^{\text{global}}$ in Eq. (2.19) of the original paper would be very small.

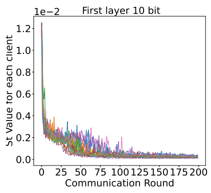
All experimental results with three different kinds of data distributions are shown in Fig. 2.6, 2.7 and 2.8, respectively. It is clear to see that, $s_t$ values from different clients do not show big differences with each other over communication rounds.



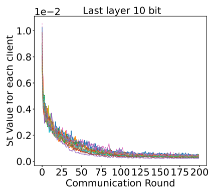
(a) First layer with 10 bit length (b) Last layer with 10 bit length (c) First layer with 15 bit length (d) Last layer with 15 bit length

Figure 2.6: $s_t$ values of all connected clients with iid data.

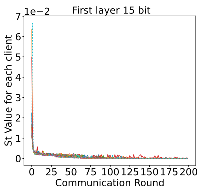
More specifically, for iid cases, the curves of different $s_t$ values are nearly located at a single line. In other words, different $s_t$ values have almost the same value and our proposed approximate aggregation algorithm has extremely small performance



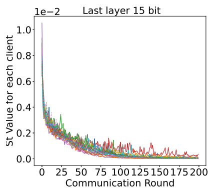
(a) First layer with 10 bit length



(b) Last layer with 10 bit length

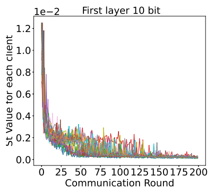


(c) First layer with 15 bit length

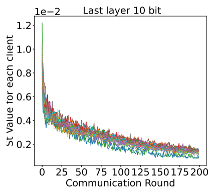


(d) Last layer with 15 bit length

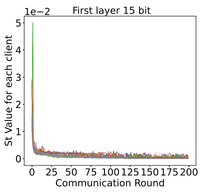
Figure 2.7: $s_t$ values of all connected clients with non iid data that each client contains only 5 different kinds of object images.



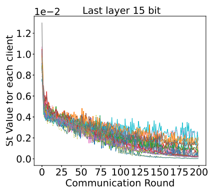
(a) First layer with 10 bit length



(b) Last layer with 10 bit length



(c) First layer with 15 bit length



(d) Last layer with 15 bit length

Figure 2.8: $s_t$ values of all connected clients with non iid data that each client contains only 2 different kinds of object images.

biases for iid data compared to original weighted averaging method.

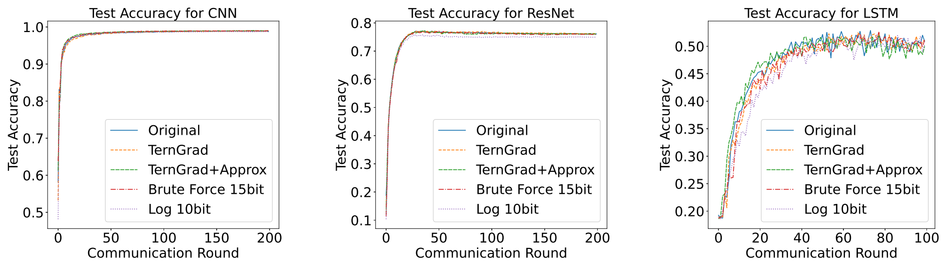
For non iid cases with 5 kinds of local images, the curves show more variations than those of iid cases. However, even for the scenarios of 10 bit encoding length, the observed maximum distance between two $s_t$ values of the first convolutional layer is less than 0.003 in Fig. 2.7(a), and the approximation biases caused by these small differences can be negligible.

The results of very extreme non iid cases with only 2 kinds of local images are also presented in Fig. 2.8. This case has an obvious difference that the $s_t$ values of the last fully connected layer show more variations than those of two previous situations. And it is also clear to see that the maximum distance between two $s_t$ values is around 0.005 in Fig. 2.8(d), thus, even for this extreme case, our proposed approximate aggregation algorithm would not bring in large performance biases and is still valid in DEAQ-FL algorithm.

## 2.4.5. Learning Performance

In this section, we empirically examine influence of the TernGrad quantization, approximate aggregation and encoding length on the learning performance of the proposed DAEQ-FL system. Fig. 2.9 shows the test accuracy of the three models with or without encryption operations. For non encryption cases, 'Original' represents standard FL, 'TernGrad' means only quantization is used and 'TernGrad+Approx' uses both quantization and approximated aggregation technique. And for encryption cases, brute force recovery is used for 15 encoding bit length cases and log recovery is adopted for 10 bit length scenarios.

From these results, we can see that the test accuracy of the models of the original FL and four variants of the DEAG-FL have achieved almost the same performance (in particular the CNN, with 98.97% test accuracy). These results indicate that both the quantization and approximated aggregation have negligible impact on the test performance of the global model. Besides, neither the quantization or encryption has considerably slowed down the convergence speed over the communication rounds. It also can be seen that, the CNN and ResNet converge around the 25th round, while the LSTM is convergent around the 50th round.

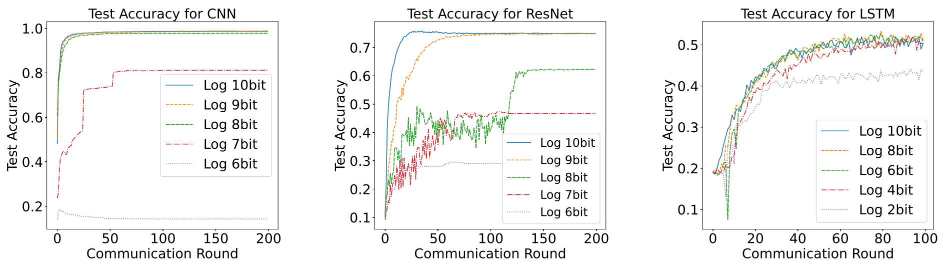


(a) CNN for MNIST (b) ResNet for CIFAR10 (c) LSTM for Shakespeare

Figure 2.9: The test accuracy of the global model for CNN, ResNet and LSTM in five different settings.

More specifically, when the CNN trained on the MNIST dataset, the model performance does not degrade when the encoding bit length is decreased to 10. However, For the ResNet trained on CIFAR10, the test accuracy of the global model is reduce to 74.96% by using 10-bit encoding, which is approximately 1% lower than models without using encryption. The test accuracy of the LSTM models fluctuates between around 48% and 52%, regardless whether quantization or encryption are used or not. This can be due to the same reasons as discussed above.

Now we take a closer look at the learning performance of the global model of the proposed DAEQ-FL when the encoding bit length further decreases. The results for all models are shown in Fig. 2.10. It can be noticed that the convergence speed of the CNN and ResNet starts to decrease when the encoding bit length is smaller than 9. Besides, the model performance reduces dramatically when the encoding length is reduced to 7 bits or lower for CNN and to 8 bits or lower for the ResNet, respectively. Reduction of the encoding bit length does not cause clear model degradation for the LSTM until when only 2 bits are used for encoding and the accuracy is reduced to 43.15%. Nevertheless, we can still observe a slight drop in the convergence speed in the beginning of the communication rounds. The possible reason for this is that the model gradients of LSTM are large.



(a) CNN for MNIST (b) ResNet for CIFAR10 (c) LSTM for Shakespeare

Figure 2.10: The test accuracy of the global model with different encoding lengths.

To take a closer look at the relationship between the model performance and computation time (related to the encoding length if the bit size is larger than 10), the test performances over different brute force recovery time are listed in Table 2.5. It is clear to see that the model test accuracy on three datasets does not have big variations with the increase of the brute force recovery time. On the CIFAR10 and MNIST datasets, however, the model performances have a significant drop from 7 bit encoding length. In this case, the log recovery method is adopted to alleviate the brute force recovery time. And all runtimes should be the same without considering the error in calculating the system computation time and other interference noise.

Overall, both TernGrad quantization and the approximate global model aggregation have little impact on the model performance, so long as the encoding bit length is not less than eight. Therefore, the proposed DEAQ-FL using the log recovery and an encoding length of 9 or 10 can achieve 128 bit security level with negligible degradation in performance and little increase in computational and communication

Table 2.5: Model test performance over different brute force recovery time on three datasets.

| Encoding bit | 6bit | 7bit | 8bit | 9bit | 10bit | 11bit | 12bit | 13bit | 14bit | 15bit |
|---|---|---|---|---|---|---|---|---|---|---|
| **CIFAR-10** | | | | | | | | | | |
| Accuracy (%) | 29.1 | 46.71 | 62.4 | 74.87 | 74.96 | 76.03 | 75.31 | 75.68 | 75.54 | 75.89 |
| Average Time (s) | \ | \ | \ | \ | \ | 0.1006 | 0.2508 | 0.5360 | 1.2207 | 2.6491 |
| Maximum Time (s) | \ | \ | \ | \ | \ | 1.4608 | 3.9389 | 7.5755 | 16.1956 | 42.2203 |
| **MNIST** | | | | | | | | | | |
| Accuracy (%) | 14.25 | 81.18 | 97.79 | 98.53 | 98.75 | 98.97 | 98.91 | 98.92 | 98.98 | 99 |
| Average Time (s) | \ | \ | \ | \ | \ | 0.1267 | 0.2312 | 0.5585 | 1.1146 | 1.9786 |
| Maximum Time (s) | \ | \ | \ | \ | \ | 1.2942 | 1.6247 | 6.3499 | 9.0804 | 18.0468 |
| **Shakespeare** | | | | | | | | | | |
| Accuracy (%) | 50.80 | 516.65 | 51.04 | 50.11 | 50.33 | 50.42 | 51.67 | 51.15 | 50.47 | 51.26 |
| Average Time (s) | \ | \ | \ | \ | \ | 1.4180 | 2.5906 | 5.4531 | 10.8060 | 23.8876 |
| Maximum Time (s) | \ | \ | \ | \ | \ | 3.2861 | 7.0456 | 11.9095 | 31.8851 | 55.2088 |

costs for the CNN, ResNet and LSTM on the corresponding datasets studied in this work.

## 2.5. Conclusion and Future Work

In this paper, we propose a privacy-preserving solution that makes use of distributed key generation and additive ElGamal encryption to protect gradients in the federated learning framework. To reduce computational and communication costs, we also introduce ternary quantization of the local models and approximate aggregation of the global model, making our solution practical in complex machine learning models, such as deep neural networks, in the context of gradient encryption. The proposed DAEQ-FL system does not rely on a TTP for key pair generations, which enables the system to tolerate a certain number of malicious clients.

DEAQ-FL can adopt the computationally efficient log recovery when the encoding bit length is less than eleven, and there is no noticeable learning performance degradation when ten bits are used for encoding (note only about 1% accuracy loss for ResNet compared to the results without doing any encryption on the Shakespeare dataset). However, the model learning performance starts to clearly deteriorate when the encoding length is less than eight. Thus, brute force recovery must be adopted when the encoding length is larger than eleven. Although a large encoding length can enhance the coding precision, the amount of recovery time will considerably increase. According to our experimental results, the global models trained with a maximum of fifteen bits can perform comparably to the non-encrypted FL, while the recovery time is still acceptable. Since the model gradients tend to decrease to zero during training, the brute force recovery time is expected to become much smaller as the global model converges.

Although the proposed method shows highly promising performance for encrypted federated deep learning, it may need to balance a trade-off between model performance and computation time needed for plaintext recovery when a small model (e.g., a logistic regression) is adopted. This is mainly because a smaller model will require a much longer encoding length (e.g., more than fifteen), making the brute force based recovery intractable. Therefore, our future work will be dedicated to the development of a distributed additive homomorphic encryption without recovery that can be used in federated learning systems.

# REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y Arcas. 'Communication-efficient learning of deep networks from decentralized data'. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.

[2] L. Li, Y. Fan, M. Tse and K.-Y. Lin. 'A review of applications in federated learning'. In: *Computers & Industrial Engineering* 149 (2020), p. 106854. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2020.106854. URL: http://www.sciencedirect.com/science/article/pii/S0360835220305532.

[3] Q. Yang, Y. Liu, T. Chen and Y. Tong. 'Federated machine learning: Concept and applications'. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2 (2019), pp. 1–19.

[4] A. Krizhevsky, I. Sutskever and G. E. Hinton. 'Imagenet classification with deep convolutional neural networks'. In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[5] Y. Chen, X. Sun and Y. Jin. 'Communication-Efficient Federated Deep Learning With Layerwise Asynchronous Model Update and Temporally Weighted Aggregation'. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.10 (2020), pp. 4229–4238. DOI: 10.1109/TNNLS.2019.2953131.

[6] H. Zhu and Y. Jin. 'Real-time Federated Evolutionary Neural Architecture Search'. In: *arXiv preprint arXiv:2003.02793* (2020).

[7] M. Ribero and H. Vikalo. 'Communication-Efficient Federated Learning via Optimal Client Sampling'. In: *arXiv:2007.15197v2* (2020).

[8] M. M. Amiri, D. Gunduz, S. R. Kulkarni and H. V. Poor. 'Federated learning with quantized global model updates'. In: *arXiv preprint arXiv:2006.10672* (2020).

[9] J. Xu, W. Du, Y. Jin, W. He and R. Cheng. 'Ternary Compression for Communication-Efficient Federated Learning'. In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[10] C. Zhu, S. Han, H. Mao and W. J. Dally. 'Trained ternary quantization'. In: *arXiv preprint arXiv:1612.01064* (2016).

[11] X. Dai, X. Yan, K. Zhou, K. K. Ng, J. Cheng and Y. Fan. 'Hyper-sphere quantization: Communication-efficient sgd for federated learning'. In: *arXiv preprint arXiv:1911.04655* (2019).

[12] Y. Du, S. Yang and K. Huang. 'High-dimensional stochastic gradient quantization for communication-efficient edge learning'. In: *IEEE Transactions on Signal Processing* 68 (2020), pp. 2128–2142.

[13] R. Shokri and V. Shmatikov. 'Privacy-preserving deep learning'. In: *the 22nd ACM CCS*.

[14] T. Orekondy, S. J. Oh, Y. Zhang, B. Schiele and M. Fritz. 'Gradient-Leaks: Understanding and Controlling Deanonymization in Federated Learning'. In: *arXiv preprint arXiv:1805.05838* (2018).

[15] L. T. Phong, Y. Aono, T. Hayashi, L. Wang and S. Moriai. 'Privacy-Preserving Deep Learning via Additively Homomorphic Encryption'. In: *IEEE Transactions on Information Forensics and Security* 13.5 (2018), pp. 1333–1345. DOI: 10.1109/TIFS.2017.2787987.

[16] J. Geiping, H. Bauermeister, H. Dröge and M. Moeller. *Inverting Gradients – How easy is it to break privacy in federated learning?* 2020. arXiv: 2003.14053 [cs.CV].

[17] H. Li and T. Han. 'An End-to-End Encrypted Neural Network for Gradient Updates Transmission in Federated Learning'. In: *2019 Data Compression Conference (DCC)*. 2019, pp. 589–589. DOI: 10.1109/DCC.2019.00101.

[18] C. Dwork. 'Differential Privacy: A Survey of Results'. In: *Theory and Applications of Models of Computation*. Ed. by M. Agrawal, D. Du, Z. Duan and A. Li. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–19. ISBN: 978-3-540-79228-4.

[19] C. Gentry and D. Boneh. *A fully homomorphic encryption scheme*. Vol. 20. 9. Stanford University Stanford, 2009.

[20] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar and L. Zhang. 'Deep learning with differential privacy'. In: *The 2016 ACM CCS*.

[21] R. C. Geyer, T. Klein and M. Nabi. 'Differentially private federated learning: A client level perspective'. In: *arXiv preprint arXiv:1712.07557* (2017).

[22] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek and H. V. Poor. 'Federated learning with differential privacy: Algorithms and performance analysis'. In: *IEEE Transactions on Information Forensics and Security* (2020).

[23] T. Yang, X. Yi, J. Wu, Y. Yuan, D. Wu, Z. Meng, Y. Hong, H. Wang, Z. Lin and K. H. Johansson. 'A survey of distributed optimization'. In: *Annual Reviews in Control* 47 (2019), pp. 278–305. ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2019.05.006. URL: https://www.sciencedirect.com/science/article/pii/S1367578819300082.

[24] X. Zhang, J. Liu, Z. Zhu and E. S. Bentley. 'Communication-Efficient Network-Distributed Optimization with Differential-Coded Compressors'. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 2020, pp. 317–326. DOI: 10.1109/INFOCOM41043.2020.9155432.

[25] S. Mao, Y. Tang, Z. Dong, K. Meng, Z. Y. Dong and F. Qian. 'A Privacy Preserving Distributed Optimization Algorithm for Economic Dispatch Over Time-Varying Directed Networks'. In: *IEEE Transactions on Industrial Informatics* 17.3 (2021), pp. 1689–1701. DOI: 10.1109/TII.2020.2996198.

[26] Y. Lu and M. Zhu. ‘Privacy preserving distributed optimization using homomorphic encryption’. In: *Automatica* 96 (2018), pp. 314–325.

[27] J. Xu, Y. Jin, W. Du and S. Gu. ‘A Federated Data-Driven Evolutionary Algorithm’. In: *arXiv preprint arXiv:2102.08288* (2021).

[28] K. Mandal and G. Gong. ‘PrivFL: Practical privacy-preserving federated regressions on high-dimensional data over mobile networks’. In: *The 2019 ACM CCSW*.

[29] M. Hao, H. Li, G. Xu, S. Liu and H. Yang. ‘Towards efficient and privacy-preserving federated deep learning’. In: *ICC 2019*. IEEE.

[30] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan and Y. Liu. ‘BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning’. In: *2020 USENIX ATC*. ISBN: 978-1-939133-14-4. URL: https://www.usenix.org/conference/atc20/presentation/zhang-chengliang.

[31] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang and Y. Zhou. ‘A Hybrid Approach to Privacy-Preserving Federated Learning’. In: *the 12th ACM AISec*. London, United Kingdom: Association for Computing Machinery. ISBN: 9781450368339. DOI: 10.1145/3338501.3357370. URL: https://doi.org/10.1145/3338501.3357370.

[32] L. Zhao, Q. Wang, Q. Zou, Y. Zhang and Y. Chen. ‘Privacy-Preserving Collaborative Deep Learning With Unreliable Participants’. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 1486–1500. DOI: 10.1109/TIFS.2019.2939713.

[33] M. Kim, J. Lee, L. Ohno-Machado and X. Jiang. ‘Secure and Differentially Private Logistic Regression for Horizontally Distributed Data’. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 695–710. DOI: 10.1109/TIFS.2019.2925496.

[34] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal and K. Seth. ‘Practical Secure Aggregation for Privacy-Preserving Machine Learning’. In: CCS ’17. 2017.

[35] K. Kursawe, G. Danezis and M. Kohlweiss. ‘Privacy-Friendly Aggregation for the Smart-Grid’. In: PETS’11. 2011.

[36] T. ElGamal. ‘A public key cryptosystem and a signature scheme based on discrete logarithms’. In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[37] I. Goodfellow, Y. Bengio, A. Courville and Y. Bengio. *Deep learning*. Vol. 1. MIT Press, Cambridge, 2016.

[38] Y. LeCun, Y. Bengio and G. Hinton. ‘Deep learning’. In: *Nature* 521.7553 (2015), pp. 436–444.

[39] J. Schmidhuber. ‘Deep learning in neural networks: An overview’. In: *Neural Networks* 61 (2015), pp. 85–117.

[40] L. Deng and D. Yu. ‘Deep learning: methods and applications’. In: *Foundations and Trends in Signal Processing* 7.3–4 (2014), pp. 197–387.

[41] M. A. Nielsen. *Neural networks and deep learning*. Vol. 2018. Determination press San Francisco, CA, 2015.

[42] Y. LeCun, Y. Bengio *et al.* ‘Convolutional networks for images, speech, and time series’. In: *The Handbook of Brain Theory and Neural Networks* 3361.10 (1995), p. 1995.

[43] M. Schuster and K. K. Paliwal. ‘Bidirectional recurrent neural networks’. In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[44] M. F. Møller. ‘A scaled conjugate gradient algorithm for fast supervised learning’. In: *Neural Networks* 6.4 (1993), pp. 525–533.

[45] P. Paillier. ‘Public-key cryptosystems based on composite degree residuosity classes’. In: *TAMC*. Springer. 1999.

[46] E. Barker, W. Barker, W. Burr, W. Polk and M. Smid. ‘Recommendation for Key Management  Part 1: General (Revision 3)’. In: *NIST Special Publication Revision* 3 (Jan. 2005).

[47] F. Knirsch, A. Unterweger, M. Unterrainer and D. Engel. ‘Comparison of the Paillier and ElGamal Cryptosystems for Smart Grid Aggregation Protocols’. In: *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP)*. Valetta, Malta: SciTePress, 2020, pp. 232–239.

[48] R. Cramer, R. Gennaro and B. Schoenmakers. ‘A secure and optimally efficient multi-authority election scheme’. In: *European Transactions on Telecommunications* 8.5 (1997), pp. 481–490.

[49] A. Shamir. ‘How to share a secret’. In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[50] P. Feldman. ‘A practical scheme for non-interactive verifiable secret sharing’. In: *SFCS 1987*. IEEE.

[51] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. ‘Secure distributed key generation for discrete-log based cryptosystems’. In: *Eurocrypt*. Springer. 1999.

[52] T. P. Pedersen. ‘Non-interactive and information-theoretic secure verifiable secret sharing’. In: *CRYTO*. Springer. 1991.

[53] X. Cao, J. Jia and N. Z. Gong. ‘Provably Secure Federated Learning against Malicious Clients’. In: *arXiv preprint arXiv:2102.01854* (2021).

[54] S. Li, Y. Cheng, W. Wang, Y. Liu and T. Chen. ‘Learning to detect malicious clients for robust federated learning’. In: *arXiv preprint arXiv:2002.00211* (2020).

[55] J.-P. Berrut and L. N. Trefethen. ‘Barycentric lagrange interpolation’. In: *SIAM Review* 46.3 (2004), pp. 501–517.

[56] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen and H. Li. ‘Terngrad: Ternary gradients to reduce communication in distributed deep learning’. In: *NIPS*. 2017.

[57] J. V. Uspensky. 'Introduction to mathematical probability'. In: (1937).

[58] L. Zhu, Z. Liu and S. Han. *Deep Leakage from Gradients*. 2019. arXiv: 1906.08935 [cs.LG].

[59] B. Zhao, K. R. Mopuri and H. Bilen. 'idlg: Improved deep leakage from gradients'. In: *arXiv preprint arXiv:2001.02610* (2020).

[60] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page and T. Ristenpart. 'Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing'. In: *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 2014, pp. 17–32.

[61] M. Fredrikson, S. Jha and T. Ristenpart. 'Model inversion attacks that exploit confidence information and basic countermeasures'. In: *the 22nd ACM CCS*.

[62] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang and H. Qi. 'Beyond inferring class representatives: User-level privacy leakage from federated learning'. In: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE. 2019, pp. 2512–2520.

[63] B. Hitaj, G. Ateniese and F. Perez-Cruz. 'Deep models under the GAN: information leakage from collaborative deep learning'. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 603–618.

[64] O. Regev. 'The learning with errors problem'. In: *Invited survey in CCC* 7.30 (2010), p. 11.

[65] Z. Brakerski, A. Langlois, C. Peikert, O. Regev and D. Stehlé. 'Classical hardness of learning with errors'. In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 2013, pp. 575–584.

[66] D. Boneh, A. Sahai and B. Waters. 'Functional encryption: Definitions and challenges'. In: *Theory of Cryptography Conference*. Springer. 2011, pp. 253–273.

[67] A. Lewko, T. Okamoto, A. Sahai, K. Takashima and B. Waters. 'Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption'. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2010, pp. 62–91.

[68] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar and H. Ludwig. 'HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning'. In: *The 12th ACM AISec*. 2019.

[69] X. Ma, F. Zhang, X. Chen and J. Shen. 'Privacy preserving multi-party computation delegation for deep learning in cloud computing'. In: *Information Sciences* 459 (2018), pp. 103–116.

[70] Y. LeCun and C. Cortes. 'MNIST handwritten digit database'. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[71] A. Krizhevsky, V. Nair and G. Hinton. 'CIFAR-10 (Canadian Institute for Advanced Research)'. In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[72] S. Hochreiter and J. Schmidhuber. 'Long short-term memory'. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[73] W. Shakespeare. ‘The Complete Works of William Shakespeare’. In: ().

[74] S. Caldas, P. Wu, T. Li, J. Konecný, H. B. McMahan, V. Smith and A. Talwalkar. ‘LEAF: A Benchmark for Federated Settings’. In: *CoRR* abs/1812.01097 (2018). arXiv: 1812.01097. URL: http://arxiv.org/abs/1812.01097.

[75] M. A. Tanner and W. H. Wong. ‘The calculation of posterior distributions by data augmentation’. In: *Journal of the American Statistical Association* 82.398 (1987), pp. 528–540.

[76] D. A. Van Dyk and X.-L. Meng. ‘The art of data augmentation’. In: *Journal of Computational and Graphical Statistics* 10.1 (2001), pp. 1–50.

[77] S. Ioffe and C. Szegedy. ‘Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift’. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[78] A. F. Agarap. ‘Deep Learning using Rectified Linear Units (ReLU)’. In: *CoRR* abs/1803.08375 (2018). arXiv: 1803.08375. URL: http://arxiv.org/abs/1803.08375.

[79] H. Zhu, H. Zhang and Y. Jin. ‘From Federated Learning to Federated Neural Architecture Search: A Survey’. In: *Complex & Intelligent Systems* 7 (2021), pp. 639–657.

[80] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

# 3

# SECURE VERTICAL FEDERATED LEARNING FOR DECENTRALIZED LABELS

*Vertical Federated Learning (VFL) enables multiple clients to collaboratively train a global model over vertically partitioned data without leaking private local information. Tree-based models, like XGBoost and LightGBM, have been widely used in VFL to enhance the interpretation and efficiency of training. However, there is a fundamental lack of research on how to conduct VFL securely over distributed labels. This work is the first to fill this gap by designing a novel protocol, called FEVERLESS, based on XGBoost. FEVERLESS leverages secure aggregation via information masking technique and global differential privacy provided by a fairly and randomly selected noise leader to prevent private information from being leaked in the training process. Furthermore, it provides label and data privacy against honest-but-curious adversaries even in the case of collusion of $n-2$ out of $n$ clients. We present a comprehensive security and efficiency analysis for our design, and the empirical results from our experiments demonstrate that FEVERLESS is fast and secure. In particular, it outperforms the solution based on additive homomorphic encryption in runtime cost and provides better accuracy than the local differential privacy approach.*

This chapter is based on the paper FEVERLESS: Fast and Secure Vertical Federated Learning based on XGBoost for Decentralized Labels by Wang, R., Ersoy, O., Zhu, H., Jin, Y. and Liang, K. IEEE Transactions on Big Data.

## 3.1. Introduction

Traditional centralized deep learning models, demanding to collect a considerable amount of clients' data to maintain high accuracy, to some degree, may increase the risk of data breaches. Data may not be easily shared among different entities due to privacy regulations and policies. To tackle this "Data Island" problem [1], Google proposed Federated Learning (FL) [2] to allow multiple clients to train a global model without sharing private data. The basic paradigm of FL is that all clients train local models with their own data, and then the information of local models, e.g., gradients, may be exchanged to produce a global model.

Based on different types of data partition [1], FL can be mainly categorized into Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL). The former focuses on training with horizontally partitioned data where clients share the same feature space but differ in data index set. Several research works [3–6] have found that training data of HFL is still at high risk of leakage although private data is kept locally. Other studies [7–11] have been dedicated to enhancing the security of HFL. On the contrary, VFL is mainly applied in the scenario of training with vertically partitioned data [12, 13] where clients share the same data index set but differ in feature space. In this paper, our principal focus is to achieve privacy-preserving training on VFL.

To the best of our knowledge, many existing studies [12–18] have proposed innovative approaches to prevent private information breaches in the context of VFL. Specifically, [14] introduced encryption-based privacy-preserving logistic regression to safeguard the information of data indexes. [15] gave a comprehensive discussion on the impact of ID resolution. [17] introduced a scheme without using a coordinator for a limited number of clients. Recently, [16] proposed an asymmetrically VFL scheme for logistic regression tackling privacy concerns on ID alignment.

Unlike the training models used in the aforementioned works, XGBoost [18], which is one of the most popular models applied in VFL, can provide better interpretation, easier parameter tuning, and faster execution than deep learning in tabular data training [19, 20]. These practical features and advantages draw academia and industry's attention to the research on XGBoost, especially in the privacy-preserving context. [12] introduced an approach for tree-based model training through a hybrid method composing homomorphic encryption and secure Multi-Party Computation (MPC) [21, 22]. After that, [13] proposed a similar system to train XGBoost [18] securely over vertically partitioned data by using Additively Homomorphic Encryption (AHE). By applying Differential Privacy (DP) [23], [24] designed a VFL system to train GBDT without the need of encryption/decryption.

However, most of the above solutions based on AHE and MPC do not scale well in terms of efficiency on training XGBoost. Beyond that, all the existing schemes basically assume that training labels are managed and processed by a *sole* client. In practice, *a VFL scheme supporting distributed labels is necessary.* For instance, multiple hospitals, clinics and health centers currently may be set to COVID-19 test spots and aim to train a model, e.g., XGBoost, to predict with good interpretation if citizens (living in various locations) are infected based on their health records and symptoms. In this context, the labels (and their values), e.g., the test results, are likely distributed among different health authorities - even targeting to the same group of patients, and feature space is vertically portioned. For example, a cardiac hospital only maintains heart data for the patients, while a psychiatric center holds the mental records, in which both authorities may collect and manage each of its registered patient's label locally. Another common scenario could be in the financial sector where multiple bank branches and e-commerce companies prefer to build a global model to predict if their customers may pay for some service (e.g., car loan) on time. The banks have part of features about the customers (e.g., account balance, funding in-and-out records), while the companies may obtain other features (e.g., payment preference). Since the customers may get the same service, e.g., loan, from different institutions, it is clear that labels must be distributed rather than centralized. In addition to efficiency and functionality aspects, one may also consider capturing stronger security for VFL. Training an XGBoost usually should involve the computation of first and second-order derivatives of the loss function (note gradients and hessians contain labels' information), and the aggregation of them is required in each round. In the context where the labels (and their values) are held by different clients, if the gradients and hessians are transmitted in the form of plaintexts and the summations of them are known to an aggregator (who could be one of the clients engages in training), inference and differential attacks (Section 3.3.3) will be easily conducted by the aggregator, resulting in information leakage.

To tackle these problems, we propose a fast and secure VFL protocol, FEVERLESS, to train XGBoost on distributed labels without disclosing both feature and label value. In our design, privacy protection is guaranteed by secure aggregation (based on a masking scheme) and Global Differential Privacy (GDP). We leverage masking instead of heavy-cost multiparty computation and we guarantee a "perfect secrecy" level for the masked data. In GDP, we use Verifiable Random Function (VRF) to select a noise leader per round (who cannot be predicted and pre-compromised in advance) to aggregate noise from "selected" clients, which significantly maintains model accuracy.

Our contributions can be summarized as follows.

(1) We define VFL in a more practical scenario where training labels are distributed over multiple clients. Beyond that, we develop FEVERLESS to train XGBoost securely and efficiently with the elegant combination of secure aggregation technique (based on Diffie-Hellman (DH) key exchange and Key Derivation Function (KDF) and GDP.
(2) We give a comprehensive security analysis to demonstrate that FEVERLESS is able to safeguard label value and feature privacy in the semi-honest setting, but also maintain robustness even for the case where $n-2$ out of $n$ clients commit collusion.
(3) We implement FEVERLESS and perform training time and accuracy evaluation on different real-world datasets. The empirical results show that FEVERLESS can maintain efficiency and accuracy simultaneously, and its performance is comparable to the baseline - a "pure" XGBoost without using any encryption and differential privacy. Specifically, training the credit card and bank marketing datasets just takes 1% and 6.5% more runtime than the baseline and meanwhile, the accuracy is only lower than that of the baseline by 0.9% and 3.21%, respectively[1].

[1]For banknote authentication dataset, FEVERLESS takes 13.96% more training time than the baseline,

## 3.2. Preliminaries

### 3.2.1. XGBoost

XGBoost [18] is a popular tree-based model in tabular data training that can provide better interpretation, easier parameters tuning and faster execution speed than deep learning [19, 20]. It also outperforms other well-known boosting tree systems in terms of accuracy and efficiency, like Spark MLLib [25] and H2O [18], especially for large-scale datasets. Therefore, in this paper, we consider using XGBoost as a building block for classification tasks.

Assume that a training set with $m$ data points composing with feature space $\mathscr{X} = \{x_1, \cdots, x_m\}$ and label space $\mathscr{Y} = \{y_1, \cdots, y_m\}$. Before training starts, every feature will be sorted based on its values, and split candidates will be set for features. XGBoost builds trees based on the determination of defined split candidates and some pruning conditions. Specifically, computing gradients and hessians first according to Eq.(3.1) and Eq.(3.2) for each data entry, where $y_i^{(t-1)}$ denotes the prediction of previous tree for $i$-th data point, and $y_i$ is the label of $i$-th data point:

$$g_i = \frac{1}{1+e^{-y_i^{(t-1)}}} - y_i = \hat{y}_i - y_i, \tag{3.1}$$

$$h_i = \frac{e^{-y_i^{(t-1)}}}{(1+e^{-y_i^{(t-1)}})^2}. \tag{3.2}$$

For splitting nodes, the XGBoost algorithm determines the best split candidate from all others based on maximum $L_{split}$ in Eq.(3.3), where $\lambda$ and $\gamma$ are regularization parameters:

$$L_{split} = \frac{1}{2}\left[\frac{\sum_{i\in I_L} g_i}{\sum_{i\in I_L} h_i + \lambda} + \frac{\sum_{i\in I_R} g_i}{\sum_{i\in I_R} h_i + \lambda} - \frac{\sum_{i\in I} g_i}{\sum_{i\in I} h_i + \lambda}\right] - \gamma. \tag{3.3}$$

The current node will be the leaf node if the following conditions are fulfilled: reaching the maximum depth of tree, the maximum value of impurity is less than a preset threshold. The calculation of the leaf value follows Eq.(3.4):

$$w = -\frac{\sum_{i\in I} g_i}{\sum_{i\in I} h_i + \lambda}. \tag{3.4}$$

### 3.2.2. Diffie-Hellman Key Exchange

Based on Decision Diffie-Hellman (DDH) hard problem [26] defined below, Diffie-Hellman key exchange (DH) [27] provides a method used for exchanging keys across public communication channels. Without losing generality and correctness, it consists of a tuple of algorithms (**Param.Gen, Key.Gen, Key.Exc**). The algorithm $(\mathbb{G}, g, q) \leftarrow$ **Param.Gen** $(1^\alpha)$ generates public parameters (a group $\mathbb{G}$ with prime order $q$ generated by a generator $g$) based on secure parameter $\alpha$. $(sk_i, pk_i) \leftarrow$ **Key.Gen**$(\mathbb{G}, g, q)$ allows client $i$ to generate secret key $(sk_i \xleftarrow{\$} \mathbb{Z}_q)$ and compute public key

and the accuracy is 30.4% lower. This is because the model is trained by a small-scale dataset, so that the robustness is seriously affected by noise.

$(pk_i \leftarrow g^{sk_i})$. Shared key is computed by $(pk_i^{sk_j}, pk_j^{sk_i}) \leftarrow \textbf{Key.Exc}(sk_i, pk_i, sk_j, pk_j)$. Inspired by [22, 28], we utilize shared keys as maskings to protect the information of labels against inference attacks during transmission in public channels. The correctness requires $pk_i^{sk_j} = pk_j^{sk_i}$. The security relies on the DDH problem [26], which is defined as:

**Definition 3.2.1** (Decision Diffie-Hellman)**.** *Let $\mathbb{G}$ be a group with prime order $q$ and $g$ be the fixed generator of the group. The Probabilistic Polynomial Time (*PPT*) adversary $\mathscr{A}$ is given $g^a$ and $g^b$ where $a$ and $b$ are randomly chosen. The probability of $\mathscr{A}$ distinguishing $(g^a, g^b, g^{ab})$ and $(g^a, g^b, g^c)$ for a randomly chosen $c$ is negligible:*

$$\Big|\Pr\Big[a, b \leftarrow\$ \mathbb{Z}_q : \mathscr{A}(g, g^a, g^b, g^{ab}) = \mathsf{true}\Big] - \Pr\Big[a, b, c \leftarrow\$ \mathbb{Z}_q : \mathscr{A}(g, g^a, g^b, g^c) = \mathsf{true}\Big]\Big| < negl(\alpha).$$

## 3.2.3. Pseudo-random Generator and Hash Function

Pseudo-Random Generator (PRG) [29] is an algorithm that is able to generate random numbers. The ”pseudo-random" here means that the generated number is not truly random but has similar properties to a random number. Generally, the pseudo-random numbers are determined by given initial values a.k.a seeds. In cryptographic applications, a secure PRG requires attackers not knowing seeds can distinguish a truly random number from an output of PRG with a negligible probability. Similar to PRG, the hash function allows mapping arbitrary sizes of data to a fixed bit value. For reducing the communication cost of FEVERLESS, we use `SHAKE-256` [30], one of the hash functions in `SHA-3` [31] family, to generate the customized size of maskings.

## 3.2.4. Key Derivation Function

Key Derivation Function (KDF) [32] is a kind of hash function that derives multiple secret keys from the main key by utilizing Pesudo-Random Function (PRF) [33]. In general, KDF algorithm $DK \leftarrow KDF(mainkey, salt, rounds)$ derives keys $DK$ based on the main key, a cryptographic salt and the current round of processing algorithm. The security requires a secure KDF that is robust for brute-force attacks or dictionary attacks. Inspired by [34] where key shares generated by DH key exchange are converted to AES keys, in this paper, we use KDF to generate maskings for every round to reduce communication costs. The main key we use is generated by DH key exchange.

## 3.2.5. Verifiable Random Function

Verifiable Random Function (VRF) [35] is a PRF providing verifiable proof of the correctness of outputs. It is a tool widely used in cryptocurrencies, smart contracts and leader selection in distributed systems [36]. Basically, given an input $x$, a signature scheme and a hash function, a practical leader selection scheme with VRF [36] works as:

$$S_{leader} \leftarrow \mathsf{H}(\mathsf{Sign}_{sk_i}(x)) \tag{3.5}$$

where $sk_i$ is the secret key for $i$-th client, and the maximum leader score $S_{leader}$ is used to determine leader. The security and unforgeability of VRF require that the signature scheme has the property of uniqueness, and the hash function is able to map the signature to a random string with a fixed size. The correctness of this $S_{leader}$ is proved by the signature of $x$.

### 3.2.6. Differential Privacy

Differential Privacy (DP) [37, 38] is a data protection system targeting on the publishing of statistical information of datasets while keeping individual data private. The security of DP requires that adversaries cannot distinguish statistical change between two datasets where an arbitrary data point is different.

The most widely used DP mechanism is called $(\epsilon,\delta)$-DP requiring less noise injected than originally proposed $\epsilon$-DP but with the same privacy level. The formal definition is given as follows.

**Definition 3.2.2.** *($(\epsilon,\delta)$ - **Differential Privacy**) Given two real positive numbers $(\epsilon,\delta)$ and a randomized algorithm $\mathscr{A}$: $\mathscr{D}^n \to \mathscr{Y}$, the algorithm $\mathscr{A}$ provides $(\epsilon,\delta)$ - differential privacy if for all data sets D, $D^{'} \in \mathscr{D}^n$ differing in only one data sample, and all $\mathscr{S} \subseteq \mathscr{Y}$:*

$$Pr[\mathscr{A}(D) \in \mathscr{S}] \leq exp(\epsilon) \cdot Pr[\mathscr{A}(D^{'}) \in \mathscr{S}] + \delta. \tag{3.6}$$

Note the noise $\mathscr{N} \sim N(0, \Delta^2\sigma^2)$ will be put into the output of the algorithm, where $\Delta$ is $l_2$ - norm sensitivity of $D$ and $\sigma = \sqrt{2\ln(1.25/\delta)}$ [39].

## 3.3. Problem Formulation

### 3.3.1. System Model

We here make some assumptions on our system. We suppose that a private set intersection [40, 41] has been used to align data IDs before the training starts, so that each client shares the same data index space $\mathscr{I}$. But the names of features are not allowed to share among clients. As for the relationship of label tagging (indexes indicating a label belongs to which client, e.g., the label $a$ is held by client $A, B$), we will consider that this can be known to the public in advance. But this assumption does not mean that the label “value" is leaked. For instance, client $C$ knows that client $A, B$ have $a$, but it does not know the specific value of $a$.

We also consider that the training is conducted on a dataset with $m$ samples composing with feature space $\mathscr{X} = \{x_1, \cdots, x_m\}$, each containing $f$ features, and label set $\mathscr{Y} = \{y_1, \cdots, y_m\}$. Besides, features $\{X_j^{(c)} \mid j \in \{1, \cdots, f\}\}$ and labels $\{y_i^{(c)} \mid i \in \{1, \cdots, m\}\}$ are held among $n$ clients where each client has at least one feature and one label.

Table 3.1: Notations summary

| **Notation** | **Description** |
|---|---|
| $\mathscr{X}$ | feature space |
| $X_j^{(c)}$ | $j$-th feature owned by $c$-th client |
| $x_i$ | $i$-th data point with $d$ features |
| $\mathscr{Y}$ | label space |
| $y_i^{(c)}$ | the label of $i$-th data point owned by $c$-th client |
| $\mathscr{I}$ | data index space |
| $\mathscr{C}$ | clients set |
| $g_i^{(c)}$ | the gradient of $i$-th data point owned by $c$-th client |
| $h_i^{(c)}$ | the hessian of $i$-th data point owned by $c$-th client |
| $G$ | summation of gradients |
| $H$ | summation of hessians |
| $m$ | number of data entries |
| $n$ | number of clients |
| $f$ | number of features |
| $d$ | the maximum depth of tree |
| $\epsilon, \delta$ | parameters of differential privacy |
| $\Delta_g$ | sensitivity of gradients |
| $\Delta_h$ | sensitivity of hessians |
| $L_{split}$ | impurity score |
| $w$ | leaf value |
| $pk_c$ | public key generated by $c$-th client |
| $sk_c$ | secret key owned by $c$-th client |
| $g$ | generator of multiplicative group |
| $B_z^j$ | $z$-th bucket of $j$-th feature |

$X_j^{(c)}$ and $y_i^{(c)}$ refer to $j$-th feature and $i$-th label owned by $c$-th client, respectively. Note we summarize the main notations in Table 3.1.

Considering a practical scenario wherein training labels are distributed among clients, we propose a new variant of VFL, named VFL over Distributed Labels (DL-VFL). The concrete definition is given as follows.

**Definition 3.3.1** (DL-VFL)**.** *Given a training set with $m$ data samples consisting of feature space $\mathscr{X}$, label space $\mathscr{Y}$ , index space $\mathscr{I}$ and clients set $\mathscr{C}$, we have:*

$$\mathscr{X}^c \cap \mathscr{X}^{c'} = \emptyset, \left| \mathscr{Y}^c \cap \mathscr{Y}^{c'} \right| < m, \mathscr{I}^c = \mathscr{I}^{c'}, \forall c, c' \in \mathscr{C}, c \neq c'.$$

*Remarks.* Different clients hold the subset of $\mathscr{X}$ sampled from feature space. A client $c$ participating DL-VFL shares the same sample ID space $\mathscr{I}$ with the corresponding labels, where a single label may be tagged to multiply clients (1-to-many case), for example, the label $a \rightarrow \mathit{client}\ A$, $B$. One may easily see the special case where a single label is assigned to only one client (i.e. $\mathscr{Y}^c \cap \mathscr{Y}^{c'} = \emptyset$), 1-to-1 case. Recall that the “tagging" relationship between label and client can be publicly known. Based on this assumption, our experiments are conducted in 1-to-1 cases for simplicity. We state that the designed experiments are also compatible with 1-to-many cases. This is so because the assumption allows the source client (which is defined below) to have knowledge of label holders, so that it can request “distinct" and missing labels from those holders, e.g., requesting $a$ from either client $A$ or $B$. Note as for 1-to-many, the size of missing labels, $|mIDs|$, could be smaller than 1-to-1, which may require less communication cost and runtime.

We will further require the participation of the source client and noise leader in our design. And they are defined as follows.

**Definition 3.3.2** (Source client). *A source client with split candidates wants to compute the corresponding $L_{split}$ based on Eq.(3.3). But some labels are missing so that $\sum g_i$ and $\sum h_i$ are unable to derive.*

For the case that a source client does not hold all labels in the current split candidates, we propose a solution based on secure aggregation and global differential privacy to help the source client to compute $L_{split}$ while safeguarding other clients' privacy. Note each client may have a chance to act as a source client because all the labels are distributed, where the *source client* leads the $L_{split}$ computation, and *clients* provide missing label values to the source client.

To achieve GDP, we define a noise leader who is selected fairly and randomly from all clients (except for the source client) - preventing clients from being compromised beforehand.

**Definition 3.3.3** (Noise leader). *By using VRF, a noise leader is responsible for generating the maximum leader score , aggregating differentially private noise from a portion of clients and adding the noise to the gradients and hessians.*

## 3.3.2. Threat Model

We mainly consider potential threats incurred by participating clients and outside adversaries. We assume that all clients are *honest-but-curious*, which means they strictly follow designed algorithms but try to infer the private information of others from the received messages. Besides, we also consider up to $n-2$ clients' collusion to conduct attacks, and at least one non-colluded client adds noise per round. Through authenticated channels, DH key exchange can be securely executed among clients. Other messages are transmitted by public channels, and outside attackers can eavesdrop on these channels and try to reveal information about clients during the whole DL-VFL process. Note this paper mainly focuses on solving privacy issues in training DL-VFL based on XGBoost. Thus, other attacks, like data poisoning and backdoor attacks deteriorating model performance, are orthogonal to our problem.

## 3.3.3. Privacy Concern

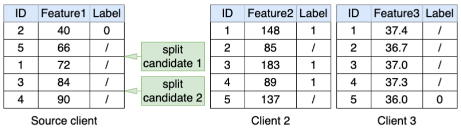
Since we assume feature names are not public information for all clients, and the values of features never leave clients, privacy issues are mainly incurred by the leakage of label information.

### INFERENCE ATTACK

During the training process, gradients and hessians are sent to the source client for $L_{split}$ computation. For the classification task, the single gradient is in the range $(-1,0) \cup (0,1)$ for binary classification. According to Eq.(3.1), a label can be inferred as 1 and 0 if the range is $(-1,0)$ and $(0,1)$, respectively. Besides, hessian illustrated in Eq.(3.2) can leak a prediction of the corresponding data sample. With training processing, the prediction is increasingly closer to a true label. The source client and outside attackers can infer the true label with high probability. Gradients and hessians cannot be transmitted in plaintext. We thus use a secure aggregation scheme to protect them from inference attacks.

### DIFFERENTIAL ATTACK

The differential attack can happen anytime and many times during the calculation of gradients and hessians. Figure 3.1 describes an example of a differential attack taking place in a single node split. After sorting feature1, the semi-honest source client defines 2 split candidates and further computes $G_{\{2,5\}} = g_2 + g_5$ and $G_{\{1,2,3,5\}} = g_2 + g_5 + g_1 + g_3$ for the candidates 1 and 2, respectively. Since the source client holds label 2, even if $G_{\{2,5\}}$ is derived by secure aggregation, the $g_5$ still can be revealed by $G_{\{2,5\}} - g_2$.



| ID | Feature1 | Label |
|---|---|---|
| 2 | 40 | 0 |
| 5 | 66 | / |
| 1 | 72 | / |
| 3 | 84 | / |
| 4 | 90 | / |

Source client

| ID | Feature2 | Label |
|---|---|---|
| 1 | 148 | 1 |
| 2 | 85 | / |
| 3 | 183 | 1 |
| 4 | 89 | 1 |
| 5 | 137 | / |

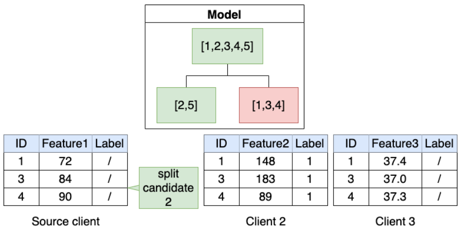
Client 2

| ID | Feature3 | Label |
|---|---|---|
| 1 | 37.4 | / |
| 2 | 36.7 | / |
| 3 | 37.0 | / |
| 4 | 37.3 | / |
| 5 | 36.0 | 0 |

Client 3

Figure 3.1: A differential attack on single node split

Another example of differential attack is shown in Figure 3.2. Assume split candidate 1 is the one for splitting the root node. In the current tree structure, the source client may split the right node by computing $L_{split}$ of split candidate 2. In this case, $G_{\{1,3\}}$ should be aggregated by the source client. And the $g_5$ can be revealed by $G_{\{1,2,3,5\}} - G_{\{1,3\}} - g_2$, where $G_{\{1,2,3,5\}}$ is computed in the previous node.



| ID | Feature1 | Label |
|---|---|---|
| 1 | 72 | / |
| 3 | 84 | / |
| 4 | 90 | / |

Source client

| ID | Feature2 | Label |
|---|---|---|
| 1 | 148 | 1 |
| 3 | 183 | 1 |
| 4 | 89 | 1 |

Client 2

| ID | Feature3 | Label |
|---|---|---|
| 1 | 37.4 | / |
| 3 | 37.0 | / |
| 4 | 37.3 | / |

Client 3

Figure 3.2: A differential attack on multiply node splits

## 3.4. A Practical Privacy-Preserving Protocol

### 3.4.1. FEVERLESS Protocol Description

To prevent a source client from knowing gradients and hessians sent by other clients, one may directly use MPC [42] based on AHE [12, 43]. But this method yields expensive computation costs. Getting rid of the complex mechanism like MPC, we leverage secure aggregation protocol via masking scheme based on DH key exchange [22, 24, 28]. By further using KDF and Hash Function, our masking (for gradients and hessians) can be derived without exchanging keys per training round. Our approach significantly reduces the communication cost but still maintains the robustness up to $n-2$ colluded clients. Meanwhile, the secure aggregation can provide “perfect secrecy" for broadcast messages. After receiving the broadcast messages, the masking will be canceled out at the source client side. But only using the masking is unable to defend against differential attacks. One may consider using Local Differential Privacy (LDP) [44] to make sure that each client may add noise per send-out message, barely consuming any extra computation cost. The accumulated noise, from all clients, may seriously affect the model's accuracy. To tackle this problem, we use a GDP [45] approach with noise leader selection. A hybrid method is finally formed based on a masking scheme and GDP, so that per client's sensitive information can be protected by the “masks" and the aggregated values are secured by the noise which is injected by the chosen clients.

We here briefly introduce our design. Assume each client $c \in [1, n]$ generates respective secret key $sk_c$ and computes gradients $g_i^{(c)}$ and hessians $h_i^{(c)}$ locally, where $\{i \mid y_i \in \mathscr{Y}^c\}$. FEVERLESS works as follows.

1. *Broadcast missing indexes.* The source client broadcasts the $mIDs = \{i \mid y_i \notin \mathscr{Y}^c\}$. Regardless of 1-to-1 or 1-to-many cases, the source client will need to send out the missing indexes (with knowledge of tagging relationships).

2. *Key exchange computation.* Each client $c$ computes public key $pk_c = g^{sk_c}$ using secret keys $sk_c$, sends $pk_c$ to other clients and computes the corresponding shared keys[2] $\{S_{c,c'} = pk_{c'}^{sk_c} = g^{sk_c sk_{c'}} \mid c, c' \in \mathscr{C}, c \neq c'\}$ based on secret key $sk_c$ received public keys $\{pk_{c'} \mid c' \in \mathscr{C}\}$.

3. *Data masking.* Each client $c$ runs the masking generation algorithm to compute the maskings for protecting gradients and hessians. Specifically, based on KDF, clients' indexes and the number of queries, the masking generation algorithm is conducted by $\mathtt{mask}_g^{(c)} \leftarrow \sum_{c \neq c'} \frac{|c-c'|}{c-c'} \cdot \left(\mathsf{H}(S_{c,c'} \| 0 \| \mathtt{query}\right)$, $\mathtt{mask}_h^{(c)} \leftarrow \sum_{c \neq c'} \frac{|c-c'|}{c-c'} \cdot \left(\mathsf{H}(S_{c,c'} \| 1 \| \mathtt{query}\right)$[3]. Then the masked gradients $G^{(c)}$ and hessians $H^{(c)}$ are generated by $G^{(c)} = \sum_{i \in mIDs} g_i^{(c)} + \mathtt{mask}_g^{(c)} - r_g^{(c)}$, $H^{(c)} = \sum_{i \in mIDs} h_i^{(c)} + \mathtt{mask}_h^{(c)} - r_h^{(c)}$.

4. *Noise leader selection.* Each client generates the selection score $selec_c$ using the VRF, $\mathsf{H}(\mathtt{SIGN}_{sk_c}(\mathtt{count}, \mathtt{mIDs}, \mathtt{r}))$, and broadcasts it, where $\mathtt{count}$ is the number of times clients conduct VRF, $\mathtt{r}$ is a fresh random number, and $\mathtt{SIGN}$ is the signature scheme. The client with the maximum score will be the noise leader. For ease of

[2]Shared keys are only generated once, and the KDF is used to generate the remaining maskings.

[3]For simplicity, we do not show the modular computations here. The full description is elaborated on Algorithm 6-8.

understanding, we assume client $n$ with the largest selection score $select_n^{max}$ is the leader, in Figure 3.3.

5. *Noise injection.* a) Noise leader selects $k$ clients adding noise. For the details of the selection, please see Algorithm 8. b) The selected clients send $\{\widetilde{n_g^{(c)}} = N(0, \Delta_g^2\sigma^2) + r_g^{(c)}, \widetilde{n_h^{(c)}} = N(0, \Delta_h^2\sigma^2) + r_h^{(c)} | c \in k\}$ to noise leader, in which the $r_g^{(c)}$ and $r_h^{(c)}$ are two random values to mask noise. c) The leader aggregates the noise: $\widetilde{N_g} = k \cdot N(0, \Delta_g^2\sigma^2) + R_g$ and $\widetilde{N_h} = k \cdot N(0, \Delta_h^2\sigma^2) + R_h$, and further adds them to $G^{(n)}$ and $H^{(n)}$, respectively.

6. *Aggregation and computation.* All clients send the masked values to the source client. The source client computes $\sum_{c=1}^{n} G^{(c)} + k \cdot N(0, \Delta_g^2\sigma^2)$, $\sum_{c=1}^{n} H^{(c)} + k \cdot N(0, \Delta_h^2\sigma^2)$ and $L_{split}$.

7. *Final update.* The source client with maximum $L_{split}$ updates the model following XGBoost [18] and broadcasts the updated model and data indexes in child nodes as step 8.

We present an overview of FEVERLESS in Figure 3.3. Note the depicted process can be conducted iteratively.

### 3.4.2. XGBOOST TRAINING OVER DISTRIBUTED LABELS

At the initial stage, we allow all clients to agree on a tree structure (maximum depth and the number of trees) and the learning rate for updating prediction. To avoid the overfitting problem, we should define regularization parameters. Threshold impurity is also another vital parameter used to identify tree and leaf nodes via the maximum impurity. After that, we should choose $\epsilon$, $\delta$ for DP, a hash function for masking generation, and noise leader selection. Besides, we select a multiplicative group $\mathbb{G}$ with order $q$ generated by a generator $g$ and a large prime number $p$ to run DH.

During the initialization process, all clients set parameters and sort their own features based on values. Then, split candidates can be defined, and data samples between two different candidates will be grouped as a bucket. In the end, all entries are assigned initialized values to calculate the derivatives of the loss function. The detailed algorithm is described as follows.

**Algorithm 4:** Initialization

```
1 Set parameters: all clients agree on the maximum depth of a tree $d$, the
    number of trees ($NT$), learning rate ($\eta$), regularization parameters ($\lambda, \gamma$), the
    threshold of $L_{split}$, $\epsilon$, $\delta$, $p$, $g$, selection portion ($p$) and hash function
2 for $c \in [1, n]$ do
3     for each feature $j$ owned by $c$ do
4         sort($X_j^{(c)}$)
5         define buckets: $B_z^j$
6     set initialized values: $\hat{y}_i^{(c)}$
```

After initialization, all clients can invoke Algorithm 5 to train the model

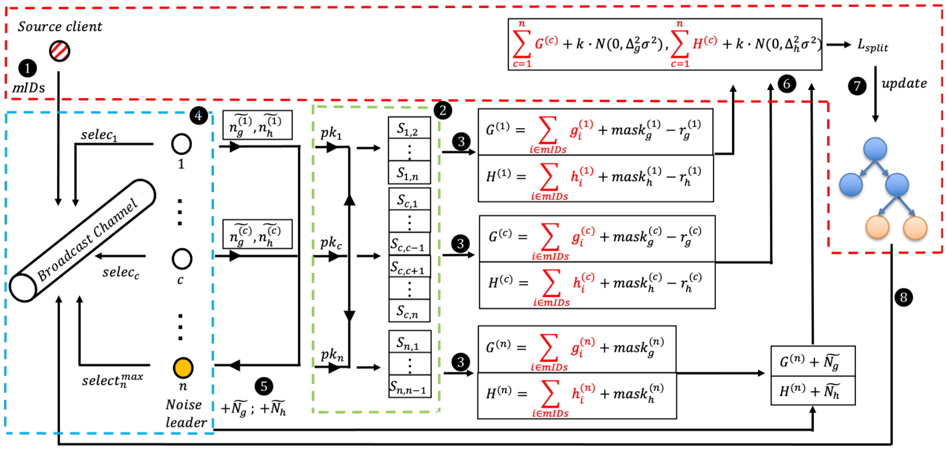


Figure 3.3: Overview of FEVERLESS. - - - - : Source client broadcasts missing *ID*s, aggregates gradients and hessians securely, updates model and broadcasts nodes *ID*s. - - - - : DH key exchange and maskings generation. - - - - : Noise leader selection. ❶Broadcast missing indexes. ❷Key exchange computation. ❸Data masking. ❹Noise leader selection. ❺Global noise injection. ❻Aggregation and computation. ❼❽ Final update and broadcast updated model. Note sensitive data are in red. The maskings in ❸ protect data from the source client, and the noise in aggregated gradients and hessians prevents the source client from conducting the differential attack.

collaboratively. The inputs are from feature space consisting of features $X_j^{(c)}$ and labels $y_i^{(c)}$ distributed on different clients, respectively; while the output is a trained XGBoost model that can be used for prediction. Generally, trees are built one by one. And we see from lines 4-10 in Algorithm 5 that each client can compute gradients and hessians at beginning of a new tree construction.

Following that, clients are to split the current node. Note that XGBoost training in DL-VFL requires each client to calculate $G$ and $H$. If the labels in some buckets are incomplete, the corresponding gradients and hessians cannot be computed. Thus, each client should first broadcast the missing data index set $mID$ (see lines 15-17 in Algorithm 5). Based on the predefined bucket $B_z^j$, $mID$ can be defined if labels in $B_z^j$ are not held by clients. In each broadcast, a client sending messages is regarded as a source client. Then others send the corresponding $g_i^{(c')}$ and $h_i^{(c')}$ back to the source client to compute $L_{split}$ through Algorithm 6-8. After finding a maximum impurity $L^c_{split\_max}$, the current node will be split into “left" and “right" nodes if $L^c_{split\_max} > threshold\_L_{split}$, in which the value of the split candidate is own by $c$. In node splitting, clients should set a given node as "leaf" if current depth reaches the predefined maximum depth or the maximum $L_{split}$ is less than the predefined threshold of $L_{split}$ (see line 12, 24-32 in Algorithm 5). The derivation of leaf value is

**Algorithm 5:** Protocol overview

```
Input: $\{X_j^{(c)} \mid j \in f, c \in |\mathscr{C}|\}$: features, $\{y_i^{(c)} \mid i \in m, c \in |\mathscr{C}|\}$: labels
Output: XGBoost model
Building trees:
for $nt \in [1, NT]$ do
    for $c \in [1, n]$ do
        for each data entry i owned by c do
            $g_i^{(c)} \leftarrow \partial_{\hat{y}_i^{(c)}} Loss(\hat{y}_i^{(c)}, y_i^{(c)})$
            $h_i^{(c)} \leftarrow \partial^2_{\hat{y}_i^{(c)}} Loss(\hat{y}_i^{(c)}, y_i^{(c)})$
        end
    end
    for each node in the current tree do
        while current depth $< d$ do
            for $c \in [1, n]$ do
                for each feature j owned by c do
                    for each $B_z^j$ owned by c do
                        Broadcast $mID = \{i \mid y_i \notin \mathscr{Y}^c\}$
                    end
                    aggregate $G$, $H$ by Algorithm 6-8
                    compute $L_{split}$ according to Eq.(3.3)
                end
                find the maximum $L_{split}^{(c)}$ and broadcast
            end
            $L_{split\_max}^{(c)} \leftarrow \max(\{L_{split}^{(c)} \mid c \in [1, n]\})$
            if $L_{split\_max}^{(c)} \leq threshold\_L_{split}$ then
                set current node as leaf node
                $c$ computes $w$ and broadcast
                Break
            else
                $c$ splits the current node to the left node and right node and
                 broadcasts the data index of them.
            end
        end
        set remaining nodes as leaf nodes
        $c$ computes $w$ and broadcast
        clients participating in calculation of $w$: update $\hat{y}_i^{(c)}$
    end
end
```

followed by Eq. 3.4 where $G$ and $H$ are intaken. Since a leaf node is either "left" or "right" split by one of the clients in $\mathscr{C}$ from its parent node, this client knows $G$

and $H$ and leaf value can be derived. Finally, this leaf value will be broadcast, and clients who own the corresponding $g_i^{(c)}$ and $h_i^{(c)}$ can use it to update predictions. The details for the above process are shown in Algorithm 5.

### 3.4.3. Secure Aggregation with Global Differential Privacy

In lines 15-19 of Algorithm 5, the source client is able to compute $L_{split}$ from the requested missing data indexes and the aggregation of received messages. To avoid that inference and differential attacks conducted on labels by source client and outside adversaries, we propose a privacy-preserving approach, shown in Algorithm 6-8, to “twist" the DH key exchange, noise leader selection and secure aggregation together. This method represents a viable alternative to train XGBoost securely in DL-VFL without demanding excessive computational resources and affecting model accuracy.

To generate the secure-but-can-be-canceled-out maskings, we adopt DH here. In Algorithm 6, all clients randomly select numbers as their secret keys and generate the corresponding public keys. For any two clients in the set $\mathscr{C}$, they will exchange the public key and compute the corresponding shared keys. For simplicity, we do not describe the signature scheme for DH. We assume DH is conducted on authenticated channels, which means the man-in-the-middle attack [46] should be invalid here.

**Algorithm 6:** Diffie-Hellman key exchange

```
1 for $c \in [1,n]$ do
2     $sk_c \leftarrow \mathbb{Z}_p^*$
3 end
4 for $c \in [1,n]$ do
5     $pk_c = g^{sk_c} \bmod p$
6     for $c' \in [1,n] \wedge c' \neq c$ do
7         $S_{c,c'} = pk_c^{sk_{c'}} \bmod p$
8     end
9 end
```

If the shared keys are used as maskings directly, our system is not robust for clients' collusion unless the amount of communication has been sacrificed as a cost to updating maskings per round. But the communication complexity is exponentially increased with the number of clients for a single node splitting. Considering the structure of trees, the overall communication complexity will be $O(2^d \cdot NT \cdot n^2)$, which may not scale well in practical applications.

To tackle this issue, we use KDF to update maskings per round automatically. Specifically, in lines 24-25 of Algorithm 8, shared keys are taken as main keys. 0 and 1 are salt values for gradients and hessians, respectively. Since the query in each round varies, the generated maskings should be dynamic accordingly. Besides, the sign of maskings is determined by the indexes of clients. In this way, we only need to use DH once, and the communication complexity is independent of tree structure.

To enable FEVERLESS to hold against differential attack, we use the GDP approach allowing the chosen one to inject a global noise to aggregated values per round. The approach is quite subtle. If the noise leader is selected by the source client, the system will be vulnerable to collusion. Moreover, a client could be easily identified as a target if we choose it in advance, e.g., by selecting a list of leaders before the training. To avoid these issues and limit the probability of collusion to the greatest extent, we use VRF to iteratively select the leader (see Algorithm 7) to securely inject a global noise. The input of VRF includes *mIDs* and a fresh random number r (line 4 in Algorithm 7), so that this client will not be predicted and set beforehand - reducing its chance to be corrupted in advance by outsiders and the source client.

**Algorithm 7:** Noise leader selection

```
1  count = 1
2  for each time run this algorithm do
3      for $c \in [1, n] \wedge c \neq source\,client$ do
4          $selec_c \leftarrow$ H(SIGN$_{sk_c}$(count,mIDs,r))
5          Broadcast
6      end
7      $selec_c^{max} \leftarrow$ max($\{selec_c \mid c \in [1, n]\}$)
8      set $c$ as noise leader
9      count+=1
10 end
```

All clients can broadcast their scores and then the one who provides the “max value" will become the leader. Then the leader re-generates a selection score as score threshold ($selec_{threshold}$) and sends it to the rest of the clients. (line 2-6 in Algorithm 8). The clients send the masked noise back to the leader if the re-generated score is larger than the threshold (lines 7-13 in Algorithm 8). Subsequently, the leader will select $\hat{k}$ clients, notify them and aggregate this masked noise to generate a global noise with a random number. In this context, even if these selected clients are colluded (note at least one is not) with the noise leader and source client, there is still a noise that cannot be recovered, safeguarding the training differentially private. Note since the noise is masked by the random number, the source client (even colluding with the leader) cannot recover the “pure" global noise to conduct the differential attack. And each client adds a noise with a probability $p$. If $k$ out of $\hat{k}$ are non-colluded, the probability of collusion is $(1-\frac{k}{n})^h$. To cancel out the randomness, the selected clients will subtract the same randomness from masked messages (line 28-31 in Algorithm 8).

Considering that the source client may procrastinate the leader selection and noise injection procedure so as to buy some time for its colluded clients to prepare sufficient large VRF values to participate in the competition of selection and adding noise. One may apply a heartbeat protocol [47] to prevent a newly elected leader from intentionally halting the noise-adding stage for a long period, say 1 min. If there is no response from the leader after for a short while, a new leader will be randomly selected. Furthermore, the heartbeat may help to solve the problem that the leader accidentally drops from the network. We note that the heartbeat protocol is not our main focus in this paper.

Before replying to the source client, we have the clients with labels put maskings on gradients and hessians, and for those without labels, they just generate and later send out maskings, in which the noise leader (i.e. one of the maskings generators) injects the noise. In this way, the maskings, guaranteeing perfect secrecy of the messages, will be canceled out after the aggregation of the values, and the differentially private noise will consolidate indistinguishability of individual data entry.

Note that in lines 24-34 of Algorithm 8, the maskings and masked values are in the range $[0, N-1]$. And $N$ should be sufficiently large to avoid overflow, and the summation of gradients and hessians should not exceed $N$.

### 3.4.4. Theoretical Analysis

**Computation cost:** We use $B$ and $d$ to denote the number of buckets and the maximum depth respectively, and $f^{(c)}$ here represents the number of features held by a client $c$. For each client $c$, the computation cost can be divided into 4 parts: (1) Performing at most $f^{(c)} \cdot B \cdot NT \cdot (2^d - 1)$ times computation of $L_{split}$ and $w$, taking $O(f^{(c)} \cdot B \cdot NT \cdot 2^d)$ time; (2) Creating $n-1$ shared keys and 1 public key, which is $O(n)$; (3) Conducting $O(f^{(c)} \cdot B \cdot NT \cdot 2^d)$ time to compute VRF outputs, select noise leader and generate noise; (4) Generating $2f^{(c)} \cdot B \cdot NT \cdot (2^d - 1)$ maskings, which takes $O(f^{(c)} \cdot B \cdot NT \cdot 2^d \cdot n)$ time. Overall, each client's computation complexity is $O(f^{(c)} \cdot B \cdot NT \cdot 2^d \cdot n)$.

**Communication cost:** Each client's communication cost can be calculated as (1) Broadcasting at most $f^{(c)} \cdot B \cdot NT \cdot (2^d - 1)$ times of missing indexes $mID$; (2) Broadcasting 1 public key and receiving $n-1$ public keys from other clients; (3) Broadcasting 1 leader selection score and sending noise to noise leader at most $f^{(c)} \cdot B \cdot NT \cdot (2^d - 1)$ times; (4) Sending source client 2 masked gradients and hessians of size $2\lceil log_2 N \rceil$. Therefore the overall communication cost is $f^{(c)} \cdot B \cdot NT \cdot (2^d - 1) \cdot (\|mID\| \cdot \alpha_I + \alpha_L + \alpha_N + n \cdot \alpha_K 2\lceil log_2 N \rceil)$, where $\alpha_I, \alpha_L$, $\alpha_N$ and $\alpha_K$ refer to the number of bits of index, leader selection score, noise and public keys, respectively. Thus, we have the communication complexity $O(f^{(c)} \cdot B \cdot NT \cdot 2^d)$.

### 3.4.5. Security Analysis

We show that FEVERLESS provides label value and data privacy against an adversary controlling at most $n-2$ clients in the *semi-honest* setting [48]. Here, we provide a brief summary of analysis and theorems.

**Label Value Privacy:** This implies that the value of a label among honest parties should not be leaked to the adversary. We achieve this by using a secure aggregation mechanism where the masks are created via DH key exchange and KDF. In brief, we show that because of the Decisional DH problem (see Definition 3.2.1), the adversary cannot distinguish the individual values from randomly chosen ones. That is why the adversary $\mathscr{A}$ cannot learn the owner of the label.

**Algorithm 8:** Secure aggregation with global differential privacy

```
1  Noise injection:
2  if $c = leader$ then
3      $selec_c^{threshold} \leftarrow \mathrm{H}(\mathtt{SIGN}_{sk_c}(\mathtt{count,mIDs,r}))$
4      Broadcast
5      count+=1
6  end
7  for $c \in [1,n] \wedge c \neq source\,client \wedge c \neq noise\,leader$ do
8      $selec_c \leftarrow \mathrm{H}(\mathtt{SIGN}_{sk_c}(\mathtt{count,mIDs,r}))$
9      if $selec_c > selec_c^{threshold}$ then
10         send $\widetilde{n_g^{(c)}} = N(0, \Delta_g^2\sigma^2) + r_g^{(c)}$ and $\widetilde{n_h^{(c)}} = N(0, \Delta_h^2\sigma^2) + r_h^{(c)}$ to noise leader
11         count+=1
12     end
13 end
14 if $c = leader$ then
15     $c$ selects $k$ clients from clients of sending noise, $k = \lceil |\{\widetilde{n_g^{(c)}}\}| \cdot p \rceil$
16     if $k < 1$ then
17         redo noise injection
18     end
19     notify $k$ clients
20     noise aggregation: $\widetilde{N_g} = k \cdot N(0, \Delta_g^2\sigma^2) + R_g$, $\widetilde{N_h} = k \cdot N(0, \Delta_h^2\sigma^2) + R_h$
21 end
22 Secure aggregation:
23 for $c \in [1,n]$ do
24     $\mathtt{mask}_g^{(c)} \leftarrow \left(\sum_{c \neq c'} \frac{|c-c'|}{c-c'} \cdot \left(\mathrm{H}(S_{c,c'} \| 0 \| \mathtt{query}) \bmod N\right)\right) \bmod N$
25     $\mathtt{mask}_h^{(c)} \leftarrow \left(\sum_{c \neq c'} \frac{|c-c'|}{c-c'} \cdot \left(\mathrm{H}(S_{c,c'} \| 1 \| \mathtt{query}) \bmod N\right)\right) \bmod N$
26     $G^{(c)} = \sum_{i \in mIDs} g_i^{(c)} + \mathtt{mask}_g^{(c)} \bmod N$
27     $H^{(c)} = \sum_{i \in mIDs} h_i^{(c)} + \mathtt{mask}_h^{(c)} \bmod N$
28     if $selec_c > selec_c^{threshold} \wedge$ received notification then
29         $G^{(c)} = G^{(c)} - r_g^{(c)} \bmod N$
30         $H^{(c)} = H^{(c)} - r_h^{(c)} \bmod N$
31     end
32     if $c = leader$ then
33         $G^{(c)} = G^{(c)} + \widetilde{N_g} \bmod N$
34         $H^{(c)} = H^{(c)} + \widetilde{N_h} \bmod N$
35     end
36     send $\{G^{(c)}, H^{(c)}\}$ to source client
37 end
```

**Data Privacy:** FEVERLESS provides data privacy, meaning that an adversary $\mathscr{A}$ cannot extract the features of training data and key shares of any honest party. Individual key shares are not separable from random values because of the secure masking. Since the calculations of gradients or hessians are irrelevant with features of training data, the adversary cannot infer features even if gradients are breached. If the source client is not part of the adversary, no data information is leaked. But we require an additional countermeasure for the case where the source client is part of the adversary because it can collect the summation of the data values. We use differential privacy [37, 38] to achieve data privacy. Because of the noise added by differential privacy, the adversary cannot learn the individual data of an honest client. Moreover, we select the noise clients by the VRF which ensures that the noise leader cannot be predicted or compromised in advance.

**Theorem 3.4.1** ($\mathscr{A}$ not including source client)**.** *There exists a Probabilistic Polynomial Time (* PPT*) simulator* Sim *for all* $|\mathscr{C}| := n \geq 3$, $|\mathscr{X}| := f \geq n$, $|\mathscr{Y}| := m \geq 1$, $\bigcup_{c\in\mathscr{C}} \mathscr{X}^{(c)}$, $\bigcup_{c\in\mathscr{C}} \mathscr{Y}^{(c)}$ *and* $\mathscr{A} \subset \mathscr{C}$ *so that* $|\mathscr{A}| \leq n-2$, *the output of* Sim *is indistinguishable from the output of* REAL: $\mathtt{REAL}_{\mathscr{A}}^{\mathscr{C},\mathscr{X},\mathscr{Y}}(\mathscr{X}^{\mathscr{C}}, \mathscr{Y}^{\mathscr{C}}) \equiv \mathsf{Sim}_{\mathscr{A}}^{\mathscr{C},\mathscr{X},\mathscr{Y}}(\mathscr{X}^{\mathscr{A}}, \mathscr{Y}^{\mathscr{A}})$.

*Proof.* In order to prove that simulator Sim can simulate the outputs of the honest parties in $\mathscr{H} := \mathscr{C} - \mathscr{A}$, we show that the distribution of the inputs belonging to the rest of the network cannot be distinguished from a randomly generated data. In this way, the simulator can use any dummy values as inputs of the honest parties to simulate their outputs.

We will simulate the view of the $\mathscr{A}$ regarding the messages broadcast by the honest clients. A client $c$, first makes a key exchange with others, then after some internal operations, outputs $G^{(c)}$ and $H^{(c)}$ values. Let us investigate $G^{(c)}$ value, which is in the form of $\sum_{i\in mIDs} g_i^{(c)} + \mathtt{mask}_g^{(c)}$, except for the noise leader who has additional noise of $N(0, (\Delta_g \sigma)^2)$. The mask values are computed as $\sum_{c\neq c'} \frac{|c-c'|}{c-c'} \cdot \mathsf{H}(S_{c,c'} \| 0 \| \mathtt{query}) \bmod N$.

Here, we will use a hybrid model where we modify the protocol in several steps, and for each step, we will show that modifications are indistinguishable for the adversary $\mathscr{A}$. In the end, we will achieve a hybrid that can be simulated by Sim.

$\mathtt{Hybrid}_1$: The first hybrid directly follows the protocol. The distribution of the variables and the view of $\mathscr{A}$ is the same as REAL.

$\mathtt{Hybrid}_2$: In the second hybrid, we replace the agreed keys between honest clients $S_{c,c'}$ for all $c, c' \in \mathscr{H}$ with random values $r_{c,c'} \in G$ where $G$ is the group of key exchange protocol $G$. In the original protocol, Diffie-Hellman key exchange is used. The replacement is indistinguishable for the adversary because of the decision Diffie-Hellman assumption given in Definition 2.1.

Also, note that these random values are only available to parties involved in the key exchange unless they are corrupted by the adversary.

$\mathtt{Hybrid}_3$: In this hybrid, we replace the mask values of honest clients $\mathtt{mask}_g^{(c)}$ for all $c \in \mathscr{H}$ with random values $R^{(c)}$. Note that with the replacement in the previous step, the mask values are computed via $\sum_{c\neq c'} \frac{|c-c'|}{c-c'} \cdot \mathsf{H}(r_{c,c'} \| 0 \| \mathtt{query}) \bmod N$ where $r_{c,c'} \in \mathscr{Z}_N$ is a random value that is unknown to the adversary (if both $c$ and $c'$ are honest). Because of the random oracle model, the output of the hash function will be a uniformly random value that is also unknown to the adversary. Since there are at most $n-2$ clients in $\mathscr{A}$ , we have at least two honest clients $c$ and $c'$ for which the adversary cannot know the uniformly chosen output of $\mathsf{H}(r_{c,c'}\|0\|\texttt{query})$. Then, the modular summation of these outputs includes at least one value that the adversary does not know and is uniformly random. Thus, it cannot be distinguishable from a random value $R^{(c)}$.

**Hybrid**$_4$: In this hybrid, we replace gradients of honest clients $g_i^{(c)}$ for all $c \in \mathscr{H}$ with '0's. This is done by replacing mask values with $\overline{R^{(c)}} := R^{(c)} - \sum_{i \in mIDs} g_i^{(c)} \bmod N$ to keep the $G^{(c)}$ value the same. From the adversary's perspective, since $R^{(c)}$ values are unknown and uniformly randomly chosen, the replacement is not distinguishable.

In **Hybrid**$_4$, we replace the gradients of honest parties with '0's, and the mask values are replaced by $\overline{R^{(c)}}$ which is unknown to the adversary and chosen from a uniform distribution. Thus, a simulator Sim can simulate the outputs of honest parties $G^{(c)}$ without necessarily knowing their inputs.

The same can be analyzed for hessian value, $H^{(c)}$. Since the masking values of $G^{(c)}$ and $H^{(c)}$ are different and the hash function is modeled as a random oracle, the randomness in both parts of them are independent of each other and indistinguishable by the adversary $\mathscr{A}$. Overall, the simulator Sim can simulate our protocol.

Thus, the view of the $\mathscr{A}$ can be simulated by replacing the inputs of the honest parties with zeros. Thus, the adversary does not learn any information on the inputs of the honest parties. $\square$

**Theorem 3.4.2** ($\mathscr{A}$ including source client)**.** *There exists a Probabilistic Polynomial Time (* PPT*) simulator* Sim *for all* $|\mathscr{C}| := n \geq 3$, $|\mathscr{X}| := f \geq n$, $|\mathscr{Y}| := m \geq 1$, $\bigcup_{c \in \mathscr{C}} \mathscr{X}^{(c)}$, $\bigcup_{c \in \mathscr{C}} \mathscr{Y}^{(c)}$ *and* $\mathscr{A} \subset \mathscr{C}$ *so that* $|\mathscr{A}| \leq n-2$, *the output of* Sim *is indistinguishable from the output of* $\texttt{REAL:REAL}_{\mathscr{A}}^{\mathscr{C},\mathscr{X},\mathscr{Y}}(\mathscr{X}^{\mathscr{C}},\mathscr{Y}^{\mathscr{C}}) \equiv \mathsf{Sim}_{\mathscr{A}}^{\mathscr{C},\mathscr{X},\mathscr{Y}}(G, H, \mathscr{X}^{\mathscr{A}}, \mathscr{Y}^{\mathscr{A}})$ *where* $G = \sum_{i \in mIDs} g_i^{(c)} + N(0, (\Delta_g \sigma)^2), H = \sum_{i \in mIDs} h_i^{(c)} + N(0, (\Delta_h \sigma)^2)$.

*Proof.* Here, we again show that Sim can simulate the outputs of the honest parties in $\mathscr{H}$ without knowing their inputs. Unlike Theorem 3.4.1, Sim is also given the summations $G$ and $H$ because the adversary includes the source client.

We can use the same hybrids with Theorem 3.4.1 until **Hybrid**$_4$, this is because that the inputs of the honest clients are not required yet. We need to update **Hybrid**$_4$ such that it takes into account the summation. Here are the hybrids for the $\mathscr{A}$ with source client:

**Hybrid**$_1$,**Hybrid**$_2$,**Hybrid**$_3$: The same with Theorem 3.4.1.

**Hybrid**$_4$: In this hybrid, we replace gradients of honest clients $g_i^{(c)}$ for all $c \in \mathscr{H}$ with '0's, except one $c'$ which will be equal to $\sum_{i \in mIDs} g_i^{(\mathscr{H})} \bmod N = G - \sum_{i \in mIDs} g_i^{(\mathscr{A})} \bmod N$. The honest client $c'$ is randomly chosen among $\mathscr{H}$. From the adversary's perspective, since $R^{(c)}$ are unknown uniformly random chosen values, the replacement is not distinguishable.

Overall, the view of the $\mathscr{A}$ can be simulated by replacing the inputs of the honest parties with zeros, except one with $\sum_{i \in mIDs} g_i^{(\mathscr{H})} \bmod N$. Thus, $\mathscr{A}$ does not learn any information from the honest clients, except the summation $\sum_{i \in mIDs} g_i^{(\mathscr{H})} \bmod N$. $\square$

With Theorem 3.4.2, we show that even the adversary $\mathscr{A}$ with source client cannot know more than the summation of gradient and hessian values, $G$ and $H$. The proof is done via Sim without requiring individual data of the honest clients except for the summation. This implies that the adversary cannot distinguish which party provided which gradient or hessian values. Moreover, the parties who do not have any of the requested $g$ or $h$ values will send '0' together with the mask (and noise for the leader). This implies that we provide *label value privacy*. Meaning that the adversary cannot distinguish which label's $g$ or $h$ values are coming from which honest client.

In the case when the adversary includes the source client, the summation of gradient and hessian values can be known to the adversary. In the following theorem, we show that these summations do not leak any individual data due to differential privacy.

**Theorem 3.4.3** (Privacy of the Inputs)**.** *No $\mathscr{A} \subset \mathscr{C}$ such that $|\mathscr{A}| \leq n-2$ can retrieve the individual values of the honest clients with probability $1-\sum_{i=0}^{\hat{k}}\binom{h}{i}\binom{n-2-h}{\hat{k}-i}(P_t)^{\hat{k}}(1-P_t)^{(n-\hat{k})}(\binom{\hat{k}-i}{k}/\binom{\hat{k}}{k})$, where h and $\hat{k}$ refer to the number of non-colluded clients and the number of clients who have selection score larger than threshold, respectively; and $P_t$ is the probability of selection score larger than the threshold.*

*Proof.* Note noise leader selects $k$ clients from $n$ clients (rather than itself and the source client) to add noise. Suppose that there are $h$ non-colluded clients out of $n-2$ clients, the number of clients whose selection scores are larger than the threshold is $\hat{k}$. The number of events is

$$C_{n-2-h}^{\hat{k}} + C_h^1 C_{n-2-h}^{\hat{k}-1} + \cdots + C_h^{\hat{k}} C_{n-2-h}^0,$$

in which the events are that {"there are $\hat{k}$ colluded clients out of $\hat{k}$ clients and 0 non-colluded client",$\cdots$,"there are 0 colluded client out of $\hat{k}$ clients and $\hat{k}$ non-colluded clients"}. Therefore,

$$\begin{aligned} P(E_i) = C_h^i (P_t)^i (1-P_t)^{h-i} \cdot \\ C_{n-2-h}^{\hat{k}-i}(P_t)^{\hat{k}-i}(1-P_t)^{(n-h-\hat{k}+i)} \\ = C_h^i C_{n-2-h}^{\hat{k}-i}(P_t)^{\hat{k}}(1-P_t)^{(n-\hat{k})}, \end{aligned}$$

where $P_t$ is the probability that the selection score is larger than the threshold, and $E_i$ is $i$-th event. Then, the probability that noise leader selects $k$ colluded clients from $\hat{k}$ clients is $P_0 = \frac{C_{\hat{k}-i}^k}{C_{\hat{k}}^k}$. At the end, the probability of all aggregated noise coming from colluded clients is

$$\begin{aligned} \sum_{i=0}^{\hat{k}} P(E_i) \cdot P_0 = \sum_{i=0}^{\hat{k}} C_h^i (P_t)^i (1-P_t)^{h-i} \cdot \\ C_{n-2-h}^{\hat{k}-i}(P_t)^{\hat{k}-i}(1-P_t)^{(n-h-\hat{k}+i)} \\ = \sum_{i=0}^{\hat{k}} C_h^i C_{n-2-h}^{\hat{k}-i}(P_t)^{\hat{k}}(1-P_t)^{(n-\hat{k})}\frac{C_{\hat{k}-i}^k}{C_{\hat{k}}^k}. \end{aligned}$$

Conversely, the probability of at least one non-colluded client participating in noise injection is

$$1-\sum_{i=0}^{\hat{k}} C_h^i C_{n-2-h}^{\hat{k}-i} (P_t)^{\hat{k}} (1-P_t)^{(n-\hat{k})} \frac{C_{\hat{k}-i}^k}{C_{\hat{k}}^k}.$$

Note that because of the secure aggregation, the adversary cannot learn anything but the summation. Thus, our protocol does not require the addition of noise to each data. Instead, we only require the noise leader to add the noise, which prevents the retrieval of the individual data from the summation. □

Note for a concrete example, if we set $n = 10, h = 2, \hat{k} = 5, k = 8, P_t = \frac{1}{2}$, the probability is 0.938. This means the source client cannot remove the noise with 0.938, which is a relatively high probability.

In Theorems 3.4.1 and 3.4.2, we show that $\mathscr{A}$ cannot distinguish the individual values from randomly chosen values and can only know the summation if the source is part of the adversary. In Theorem 3.4.3, we show that $\mathscr{A}$ cannot extract the individual values of the users from the summation due to the added noise and differential privacy. Thus, our protocol satisfies *data privacy*. In other words, the adversary cannot learn the data point of an honest client.

It is important to note that since the noise leader is selected via VRF, no adversary can guess if any honest party will be the leader in the upcoming round beforehand. This provides additional security regarding the manipulation of the noise leader.

## 3.5. EXPERIMENT

We perform evaluations on the accuracy, runtime performance and communication cost, and compare our design with two straightforward secure approaches: one is based on LDP (for accuracy), and the other is built on AHE with GDP (for runtime). These approaches are most-commonly-used components for privacy-preserving FL, and they could be the building blocks for complex mechanisms, e.g., MPC. We note the protocol should intuitively outperform those MPC-based solutions, and one may leverage our source code to make further comparisons if interested. In the experiments, the baseline, which is the pure XGBoots algorithm, follows the training process of Figure 3.3 without using any privacy-preserving tools (steps ❷ - ❺). And LDP does not conduct DH key exchange but each client injects noise into the aggregation of gradients and hessians, while AHE follows Figure 3.3 except executing DH key exchange. In AHE, each client sends (additive) encrypted messages to the source client after step ❺. We here show the performance of the best case where there is only one (non-colluded and randomly selected) client adding noise per round ($k = 1$).

### 3.5.1. EXPERIMENT SETUP

All the experiments are implemented in Python, and conducted on a cluster of machines with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz, with 15GB RAM in a

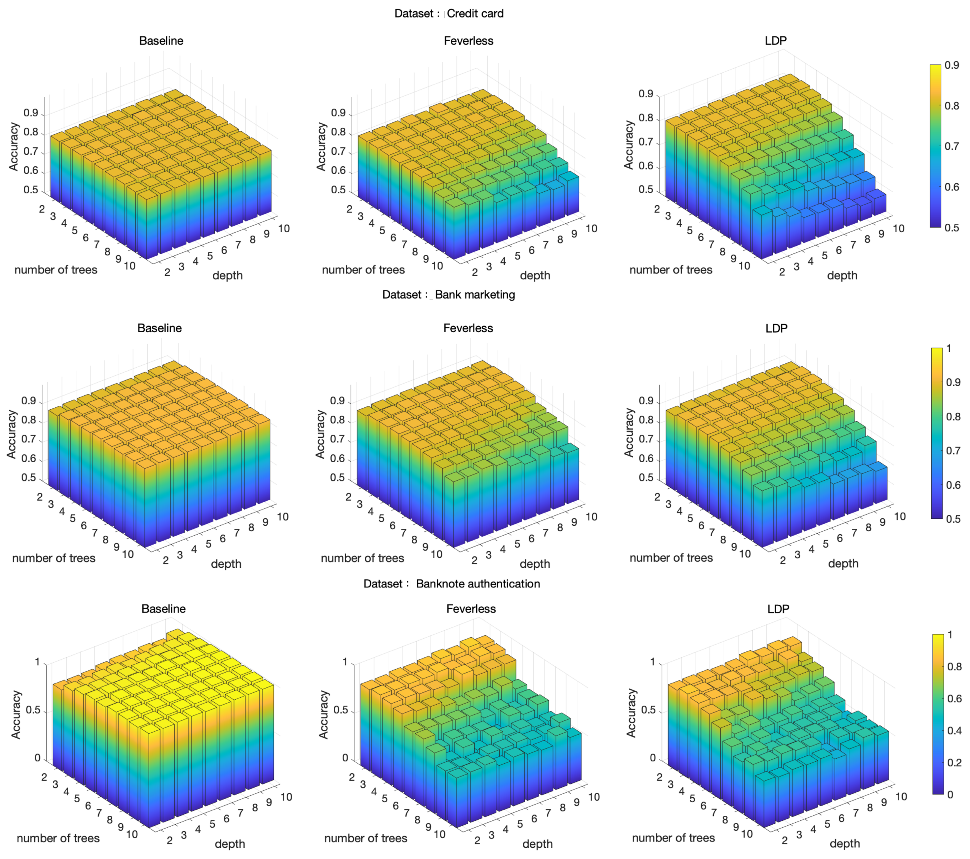


Figure 3.4: Comparison among the baseline, FEVERLESS and LDP under $\epsilon = 2$. *Top row:* Credit card dataset, accuracy range: [0.5, 0.9]. *Middle row:* Bank marketing, accuracy range: [0.5, 1]. *Bottom row:* Banknote authentication, accuracy range: [0, 1].

local area network. As for the cryptographic tools, we set the key size of DH and Paillier as 160 bits and 1024 bits respectively(to save some time in running the experiments). This size can reach a symmetric security level with 80 bits key length. Note one may indeed increase the key size to obtain stronger security [4], but this will bring a longer experiment time as a side effect. We use 1024-bit MODP Group with 160-bit Prime Order Subgroup from *RFC 5114* [5] for DH Key exchange. `SHAKE-256` [49], a member of `SHA3` [49] family, is used as a hash function in leader selection and secure aggregation.

Intuitively, the smaller $\epsilon$ we set, the more secure FEVERLESS will be; but larger noise will be added. We note the above statement can be seen from the experimental

[4]Note a stronger security level will not affect the training accuracy.
[5]https://tools.ietf.org/html/rfc5114

results. To present comprehensive results on the accuracy, we set $\epsilon$ to be: 10, 5, 2 and 1, and $\delta$ is set to $10^{-5}$. In terms of accuracy and runtime, we evaluate different situations by varying the number of clients, the number of trees, and the maximum depth of trees (from 2 to 10). Other parameters regarding training follow the suggestions in [18] and the *library* [6] of XGBoost. To deliver fair results, we conduct each test for 20 independent trials and then calculate the average.

**Datasets.** We run the experiments on three datasets - Credit Card [50], Bank Marketing [51] and Banknote Authentication[7] - for classification tasks. Because the more concentrated the distribution of labels and features, the more like centralized learning. The entire algorithm requires less interaction among clients. This situation is less common in practical applications. To fairly investigate the model performance in DL-VFL, we make the features and labels as sparse as possible, and they are uniformly distributed among clients.

- **Credit Card:** It is a commercial dataset used for predicting whether customers will make payments on time. It provides 30,000 samples, and each sample composes of 23 features.
- **Bank marketing:** Consisting of 45,211 data points and 17 features, the goal of bank marketing is to predict if a client will subscribe to a term deposit.
- **Banknote authentication:** Offering 1,372 data points and 4 features, this dataset is used to classify authenticated and unauthenticated banknotes. Note that different from traditional tabular data, features in the dataset are extracted from images that are taken from genuine and forged banknote-like specimens through Wavelet Transform [52]. Using the small-scale dataset, the trained model may not be robust to noise, which brings a negative impact on accuracy.

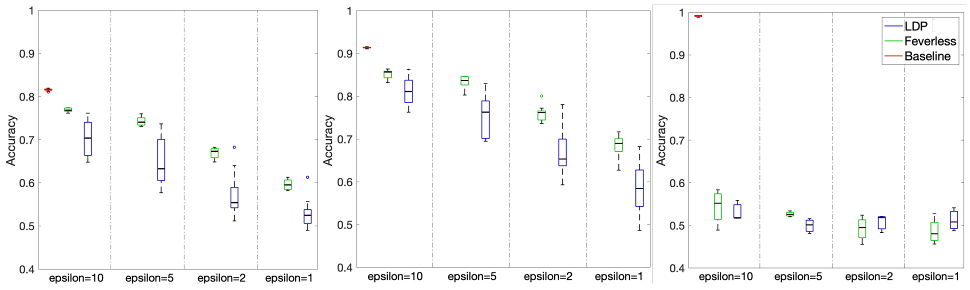


Figure 3.5: Comparison of accuracy by varying $\epsilon$ in depth=10, the number of trees=10. *Left:* Credit card. *Middle:* Bank marketing. *Right:* Banknote authentication. Accuracy ranges from 0.4 to 1.

## 3.5.2. EVALUATION ON ACCURACY

In Figure 3.4, we present a clear picture of the accuracy performance based on the #tree and the maximum depth in $(2, 10^{-5})-$DP. We merge the #client in one tree

[6]https://xgboost.readthedocs.io/
[7]https://archive.ics.uci.edu/ml/datasets/banknote+authentication

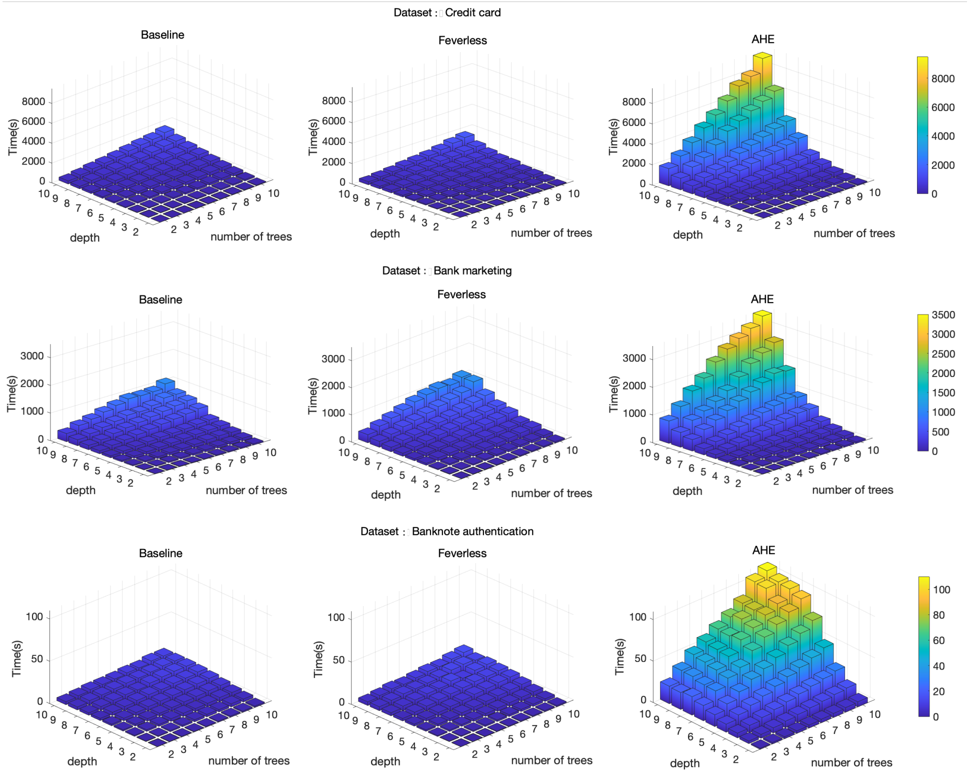


Figure 3.6: Comparison of time. *Top row:* Credit card dataset, range: [0s, 9,500s]. *Middle row:* Bank marketing, range: [0s, 3,500s]. *Bottom row:* Banknote authentication, range: [0s, 110s].

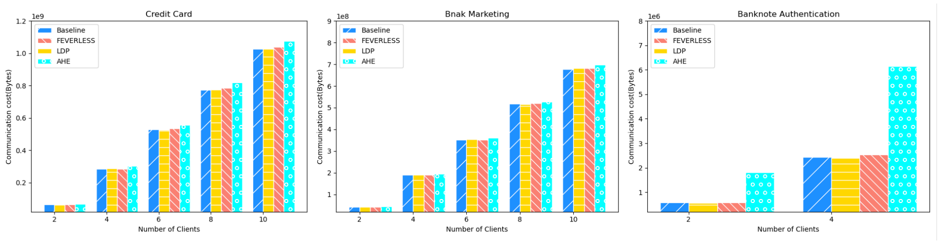


Figure 3.7: Comparison of communication cost on the number of clients.

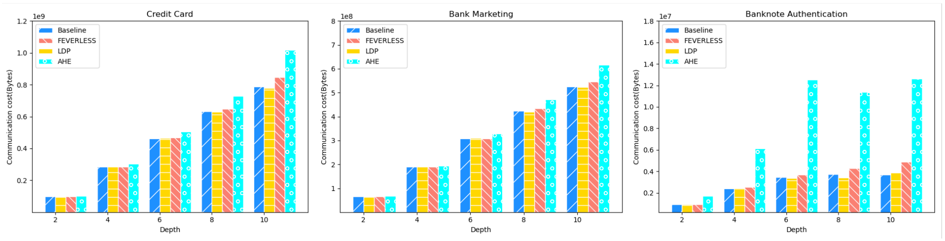


Figure 3.8: Comparison of communication cost on depth.

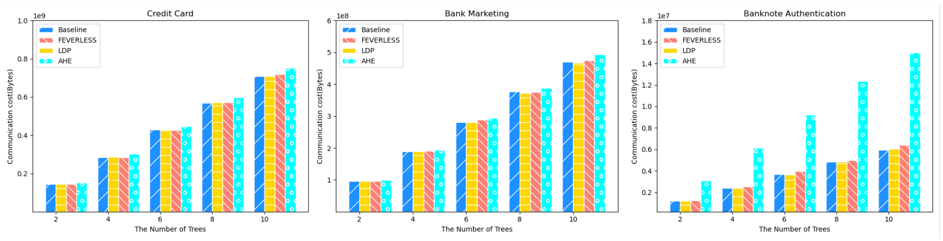


Figure 3.9: Comparison of communication cost on the number of trees.

structure, which means in one bar, and the value is the mean of accuracy when conducting on different numbers. The accuracy of the baseline in credit card (about 0.82) and bank marketing (nearly 0.9) remains unchanged as the #tree and maximum depth increases, while the accuracy in banknote authentication rises from 0.9 to approximately 1.0. To highlight the differences and ensure all results are displayed clearly, we set the ranges of accuracy as $[0.5, 0.9], [0.5, 1]$ and $[0, 1]$ for the three datasets, respectively.

Compared with the baseline, shown in the top and middle rows of Figure 3.4, FEVERLESS and LDP suffer from continuously shrinking accuracy as tree structure becomes complex. This is so because the injected noises are accumulated into the model via the increase of query number. And the accuracy is easily affected by the depth. In the worst case where the #tree and maximum depth are both equal to 10, FEVERLESS decreases 10.37% (resp. 14.98% ), and LDP drops 24.78% (resp.24.59%) in credit card (resp. bank marketing). But on average, FEVERLESS' accuracy only shrinks by around 0.9% (resp. 3.21%), while LDP suffers from an estimated 3x (resp. 2x) accuracy loss. The difference in the degree of deterioration mainly comes from how much noise is added for each query. We note the deterioration of FEVERLESS is independent of the #client. Thus, we can maintain great accuracy even in the case where there exists a considerable amount of clients.

Despite the fact that less noise is added in FEVERLESS, we do not predict that the accuracy falls to the same level (around 50%, like randomly guessing in binary classification) as LDP in the bottom row of Figure 3.4. This is so because the model is trained by an extremely small-size dataset, which makes it hard to maintain the robustness but relatively sensitive to noise. If setting a larger $\epsilon$, we may see our advantage more clearly.

To distinguish the performance between FEVERLESS and LDP more clearly, Figure 3.5 shows the comparison over different $\epsilon$, when #depth and #tree are set to 10. The performance of the model is decayed as the decrease of $\epsilon$. In the left (resp. middle) of Figure 3.5, the averaged accuracy of FEVERLESS falls from 0.7686 to 0.5967 (resp. from 0.8517 to 0.6831), while that of LDP also decreases to 0.5299 (resp. 0.5853). We notice that the highest values of LDP stay at the same level as those of FEVERLESS. This is because, in the case of 2-client training, only one client needs to add the noise in LDP (which is identical to our GDP solution). At last, the worse case can be seen on the right of Figure 3.5 due to the weak robustness of the model obtained from the banknote authentication. The results are far away from the baseline there. This is because in small-scale datasets, the heterogeneity of data distribution is not large, so the original XGBoost can achieve high accuracy. However, the model trained in this way is less robust, which means it is more sensitive to noise. Therefore, compared with the model trained on a large-scale dataset, it does not perform well under the condition of differential privacy. But even in this case, FEVERLESS still holds a tiny advantage over LDP.

Note that we did not compare the accuracy to systems using AHE. Because the calculation process of homomorphic encryption does not change the precision of the value, training through encryption will not affect the model. Therefore, the accuracy of using AHE is the same as the pure XGBoost.

### 3.5.3. Evaluation on Training Time

To highlight the runtime complexity, we average the results varying by client number into one tree structure as well. We further set the ranges of time as [0s, 9,500s], [0s, 3,500s] and [0s, 110s] for the datasets to deliver visible results. Note since the banknote dataset contains the least samples, it does deliver the best training efficiency here. Figure 3.6 presents the comparison of the training time by varying maximum depth and the number of trees among the datasets.

The training time increases exponentially and linearly with depth and the number of tree, which is consistent with our analysis given in Section 3.4.4. In Figure 3.6, compared with the baseline, the runtime of FEVERLESS at most increases 110.3s (resp. 50s, 4.3s), while AHE requires around 70x spike (resp. 48x, 21x) in credit card (resp. bank marketing, banknote authentication), where #depth and #trees are equal to 10. For the average case, FEVERLESS consumes Approx.

1%($resp.$6.5%, 13.96%) more training time than the baseline, while AHE requires the 351%($resp.$155.1%, 674%) extra, w.r.t. the three datasets. Its poor performances are due to the laborious calculations in encryption, in which each client has to conduct an encryption per query. By contrast, the maskings in FEVERLESS avoid these excessive costs. In Figure 3.10-3.14, we show the time performance based on various numbers of client, tree and depth. In general, the runtime of FEVERLESS is slightly higher that that of the baseline. Compared to AHE, FEVERLESS significantly reduces training time while preserving privacy. This advantage is clearly seen from the cases using complex tree structures. Note that AHE can be replaced by other more complex cartographic solutions, such as secure MPC, which can also maintain data/label privacy. But the MPC-based solutions will consume more runtime.

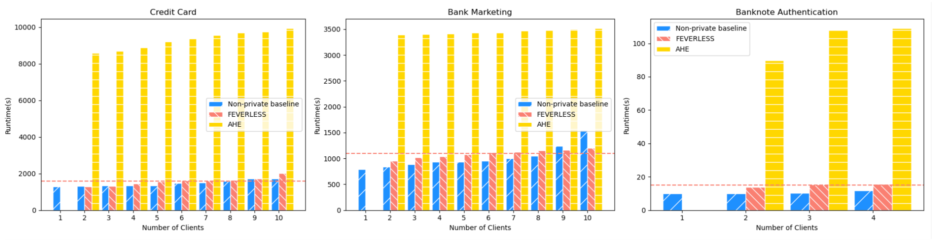


Figure 3.10: Comparison of runtime in depth:10, the number of trees:10.

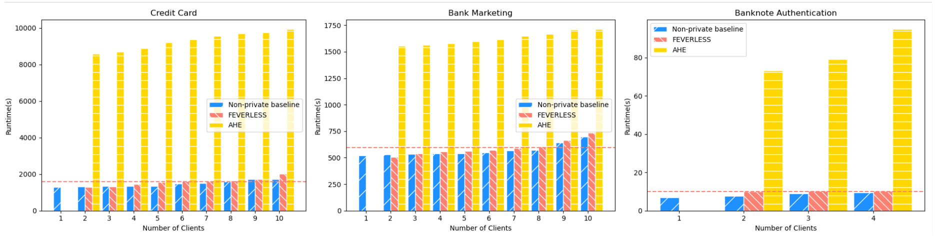


Figure 3.11: Comparison of runtime in depth:8, the number of trees:8.

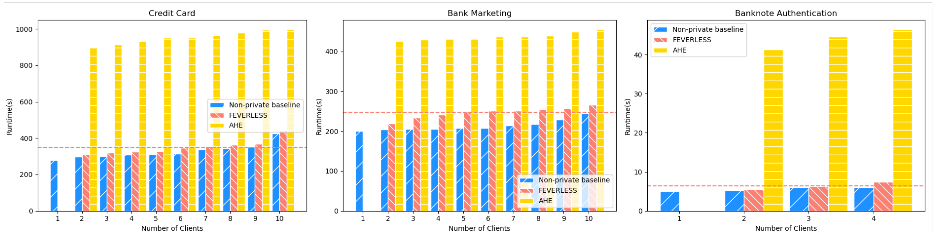


Figure 3.12: Comparison of runtime in depth:6, the number of trees:6.

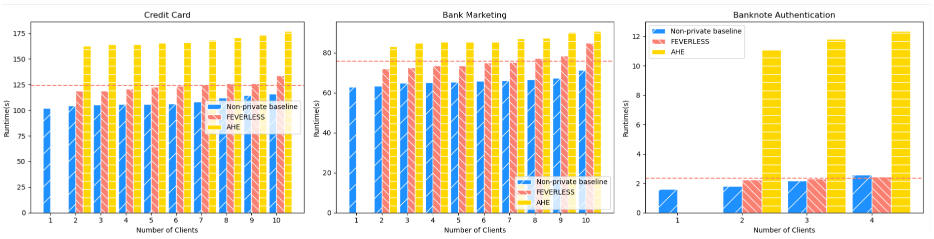


Figure 3.13: Comparison of runtime in depth:4, the number of trees:4.

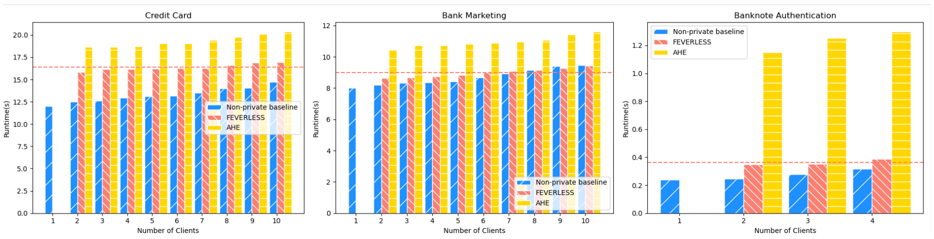


Figure 3.14: Comparison of runtime in depth:2, the number of trees:2.

## 3.5.4. Evaluation on Communication Cost

In Figure 3.7-3.9, we demonstrate the communication cost based on the number of clients, tree and depth. For the convenience of comparison, we set #clients=4, #tree=4 and depth=4 as default. To sum up, we see that the communication cost of FEVERLESS is almost the same as those of the baseline and LDP. But as compared to AHE, FEVERLESS significantly reduces the cost while maintaining privacy.

In each presented figure, we show the results executed on the datasets Credit card (left), Bank Marketing (middle) and Banknote Authentication (right). Note that the comparison among FEVERLESS, LDP, and AHE requires a condition that #client=2; when #client=1, we can only show the results of the baseline.

Via the experiments, we elaborate that how the communication cost varies with the increasing number of clients, depth and the number of trees among the baseline, FEVERLESS, AHE and LDP.

In general, adding noise has no clear impact on communication costs. The performance of FEVERLESS and LDP is on par with that of the baseline. The AHE approach does harm communication costs, which can be seen from the continuously and significantly increasing bars in the figures. Naturally, when more clients engage in the training, more communication costs should be added to the model. Especially, in the number of clients equal to 4, the communication costs of AHE is around 6*1e6 Bytes in Banknote Authentication dataset, which is about 3x than other methods. Similar situations can be observed when training with complex tree structures. In depth (*resp.* the number of trees) equals 10, the communication costs of AHE reaches about 1.3*1e7 Bytes (*resp.* 1.5*1e7 Bytes), which is 2.6x (*resp.* 2.4x) than other methods. AHE generates such a large amount of communication costs because it requires transmitting ciphertexts during interactions among clients.

## 3.6. DISCUSSION

To reduce the negative impact brought by noise, according to infinity divisibility of Gaussian distribution [53], one may split global noise $(N(0,(\Delta\sigma)^2))$ into $n$ parts $(N(0,\frac{(\Delta\sigma)^2}{n}))$. But a drawback is that the privacy budget will increase linearly as an increasing number of colluded clients appear. For example, if GDP achieves $\epsilon$-$DP$, in the worst case where there are $n-1$ colluded clients, the privacy budget will raise to $n\times\epsilon$.

**Hiding label tagging information**. In the semi-honest setting, if the source client sends the missing indexes consistently, adversaries may figure out which labels are distributed (on the source clients) by statistical analysis. We show that this issue can be tackled in 1-to-1 case. In the proposed protocol, source client broadcasts the missing data indexes $mID$). Under the semi-honest setting, if source client sends missing indexes consistently, the adversaries will figure out which labels are distributed on source clients by statistic analysis. We note that FEVERLESS can be expanded to avoid this type of leakage by yielding extra communication overheads. Specifically, during broadcasting period, source client should send indexes of one bucket instead of $mID$, and the rest of protocol remains constant. In this way, others cannot distinguish the distribution of labels because all clients share the same index set $\mathscr{I}$. If we assume labels are uniformly distributed on each client, the extra overheads are restricted to $|\mathscr{I}|/|\mathscr{C}|$. This cost is clearly noticeable in those datasets with a large number of data points.

Note that the solution described above does not work for the 1-to-many case. If the source client does not know the label tagging relationship, other clients holding the same labels will be all required to send their masked gradients and hessians after receiving the $mIDs$. These repeated values may lead to incorrect calculations of $L_{split}$. We leave this as an interesting open problem.

**Other security tools**. The masking scheme realizing secure aggregation may be replaced with an MPC [12, 42] or additively homomorphic encryption [43]. However, the major defect of these tools is that they entail labor-intensive calculation with regard to encryption, which may not scale well in large-scale datasets. Due to this concern, we only put light-weight computation in FEVERLESE and further, we enhance the security to “perfect secrecy".

In our design, the selection of noise leader is captured by VRF. We note that there may be other options to fulfill the goal. For example, Proof of Elapsed Time (PoET) [54, 55] is an interesting and effective mechanism which is used to maintain the consensus of distributed peers in Hyperledger Sawtooth. It provides a fair and trusted lottery strategy to select a block winner (per consensus round). Sharing the same philosophy with the VRF, it may be deployed in our protocol to yield leader. And building a more efficient noise leader selection algorithm could be an interesting open problem.

## 3.7. Conclusion and Future Work

We consider a practical scenario where labels are distributedly and maintained by different clients for VFL. By leveraging secure aggregation and GDP, we present a novel system, FEVERLESS, to train XGBoost securely. FEVERLESS can achieve perfect secrecy for labels and data, and adversaries cannot learn any information about the data even if the source client is corrupted. With DP against differential attack, the source client knows nothing more than summation. Our design is also robust for the collusion of $n-2$ out of n clients. FEVERLESS is about the same speed and accuracy as the pure XGBoost, taking 1% extra runtime, and sacrificing 0.9% accuracy. Although our system achieves great performance in terms of security and efficiency, its accuracy still does not work well in small-scale datasets. This remains an open problem. We will also consider secure solutions against malicious adversaries.

# REFERENCES

[1] Q. Yang, Y. Liu, T. Chen and Y. Tong. 'Federated Machine Learning: Concept and Applications'. In: *ACM Trans. Intell. Syst. Technol.* 10.2 (Jan. 2019).

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y Arcas. 'Communication-efficient learning of deep networks from decentralized data'. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.

[3] R. Shokri and V. Shmatikov. 'Privacy-Preserving Deep Learning'. In: CCS '15, pp. 1310–1321.

[4] T. Orekondy, S. J. Oh, Y. Zhang, B. Schiele and M. Fritz. 'Gradient-Leaks: Understanding and Controlling Deanonymization in Federated Learning'. In: *NeurIPS Workshop on Federated Learning for Data Privacy and Confidentiality*. 2019.

[5] J. Geiping, H. Bauermeister, H. Dröge and M. Moeller. 'Inverting Gradients - How easy is it to break privacy in federated learning?' In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 16937–16947. URL: https://proceedings.neurips.cc/paper/2020/file/c4ede56bbd98819ae6112b20ac6bf145-Paper.pdf.

[6] H. Li and T. Han. 'An end-to-end encrypted neural network for gradient updates transmission in federated learning'. In: *arXiv preprint arXiv:1908.08340* (2019).

[7] L. T. Phong, Y. Aono, T. Hayashi, L. Wang and S. Moriai. 'Privacy-Preserving Deep Learning via Additively Homomorphic Encryption'. In: *IEEE Transactions on Information Forensics and Security* 13.5 (2018), pp. 1333–1345. DOI: 10.1109/TIFS.2017.2787987.

[8] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang and Y. Zhou. 'A Hybrid Approach to Privacy-Preserving Federated Learning'. In: *the 12th ACM AISec*. London, United Kingdom: Association for Computing Machinery. ISBN: 9781450368339. DOI: 10.1145/3338501.3357370. URL: https://doi.org/10.1145/3338501.3357370.

[9] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar and H. Ludwig. 'HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning'. In: *The 12th ACM AISec*. 2019.

[10] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan and Y. Liu. 'BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning'. In: *2020 USENIX ATC*. ISBN: 978-1-939133-14-4. URL: https://www.usenix.org/conference/atc20/presentation/zhang-chengliang.

[11] H. Zhu, R. Wang, Y. Jin, K. Liang and J. Ning. ‘Distributed Additive Encryption and Quantization for Privacy Preserving Federated Deep Learning’. In: *arXiv preprint arXiv:2011.12623* (2020).

[12] Y. Wu, S. Cai, X. Xiao, G. Chen and B. C. Ooi. ‘Privacy Preserving Vertical Federated Learning for Tree-Based Models’. In: *Proc. VLDB Endow.* (2020), pp. 2090–2103.

[13] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos and Q. Yang. *SecureBoost: A Lossless Federated Learning Framework*. 2021. arXiv: 1901.08755 [cs.LG].

[14] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith and B. Thorne. ‘Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption’. In: *arXiv preprint arXiv:1711.10677* (2017).

[15] R. Nock, S. Hardy, W. Henecka, H. Ivey-Law, G. Patrini, G. Smith and B. Thorne. ‘Entity resolution and federated learning get a federated resolution’. In: *arXiv preprint arXiv:1803.04035* (2018).

[16] Y. Liu, X. Zhang and L. Wang. ‘Asymmetrically vertical federated learning’. In: *arXiv preprint arXiv:2004.07427* (2020).

[17] S. Yang, B. Ren, X. Zhou and L. Liu. ‘Parallel distributed logistic regression for vertical federated learning without third-party coordinator’. In: *arXiv preprint arXiv:1911.09824* (2019).

[18] T. Chen and C. Guestrin. ‘XGBoost: A Scalable Tree Boosting System’. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785.

[19] I. Goodfellow, Y. Bengio, A. Courville and Y. Bengio. *Deep learning*. Vol. 1. MIT Press, Cambridge, 2016.

[20] Y. LeCun, Y. Bengio and G. Hinton. ‘Deep learning’. In: *Nature* 521.7553 (2015), pp. 436–444.

[21] O. Goldreich. ‘Secure multi-party computation’. In: *Manuscript. Preliminary version* 78 (1998).

[22] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal and K. Seth. ‘Practical Secure Aggregation for Privacy-Preserving Machine Learning’. In: CCS ’17. 2017.

[23] C. Dwork. ‘Differential Privacy: A Survey of Results’. In: *Theory and Applications of Models of Computation*. Ed. by M. Agrawal, D. Du, Z. Duan and A. Li. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–19. ISBN: 978-3-540-79228-4.

[24] Z. Tian, R. Zhang, X. Hou, J. Liu and K. Ren. *FederBoost: Private Federated Learning for GBDT*. 2020. arXiv: 2011.02796 [cs.CR].

[25] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.* 'Mllib: Machine learning in apache spark'. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1235–1241.

[26] D. Boneh. 'The Decision Diffie-Hellman Problem'. In: *ANTS*. Vol. 1423. Lecture Notes in Computer Science. Springer, 1998, pp. 48–63.

[27] W. Diffie and M. Hellman. 'New directions in cryptography'. In: *IEEE transactions on Information Theory* 22.6 (1976), pp. 644–654.

[28] G. Ács and C. Castelluccia. 'I Have a DREAM! (DiffeRentially privatE smArt Metering)'. In: *Information Hiding*. 2011.

[29] J. Håstad, R. Impagliazzo, L. A. Levin and M. Luby. 'A pseudorandom generator from any one-way function'. In: *SIAM Journal on Computing* 28.4 (1999), pp. 1364–1396.

[30] N. Sha. *standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication 202, 2015*. 2015.

[31] J.-P. Aumasson, L. Henzen, W. Meier and R. C.-W. Phan. 'Sha-3 proposal blake'. In: *Submission to NIST* 92 (2008).

[32] H. Krawczyk and P. Eronen. *HMAC-based extract-and-expand key derivation function (HKDF)*. Tech. rep. RFC 5869, May, 2010.

[33] B. Kaliski. 'Pseudorandom Function'. In: *Encyclopedia of Cryptography and Security*. Ed. by H. C. A. van Tilborg. Boston, MA: Springer US, 2005, pp. 485–485. ISBN: 978-0-387-23483-0. DOI: 10.1007/0-387-23483-7_329. URL: https://doi.org/10.1007/0-387-23483-7_329.

[34] J. Zdziarski. *Hacking and securing iOS applications: stealing data, hijacking software, and how to prevent it*. " O'Reilly Media, Inc.", 2012.

[35] S. Micali, M. Rabin and S. Vadhan. 'Verifiable random functions'. In: *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE. 1999, pp. 120–130.

[36] S. Micali. 'ALGORAND: The Efficient and Democratic Ledger'. In: *CoRR* abs/1607.01341 (2016). arXiv: 1607.01341. URL: http://arxiv.org/abs/1607.01341.

[37] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov and M. Naor. 'Our data, ourselves: Privacy via distributed noise generation'. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2006, pp. 486–503.

[38] C. Dwork, F. McSherry, K. Nissim and A. Smith. 'Calibrating noise to sensitivity in private data analysis'. In: *Theory of cryptography conference*. Springer. 2006, pp. 265–284.

[39] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar and L. Zhang. 'Deep learning with differential privacy'. In: *The 2016 ACM CCS*.

[40] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek and N. Trieu. ‘Practical multi-party private set intersection from symmetric-key techniques’. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1257–1272.

[41] B. Pinkas, T. Schneider and M. Zohner. ‘Faster private set intersection based on {OT} extension’. In: *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 2014, pp. 797–812.

[42] I. Damgård, V. Pastro, N. Smart and S. Zakarias. ‘Multiparty computation from somewhat homomorphic encryption’. In: *Annual Cryptology Conference*. Springer. 2012, pp. 643–662.

[43] P. Paillier. ‘Public-key cryptosystems based on composite degree residuosity classes’. In: *TAMC*. Springer. 1999.

[44] P. Kairouz, S. Oh and P. Viswanath. ‘Extremal mechanisms for local differential privacy’. In: *Advances in neural information processing systems* 27 (2014), pp. 2879–2887.

[45] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek and H. V. Poor. ‘Federated learning with differential privacy: Algorithms and performance analysis’. In: *IEEE Transactions on Information Forensics and Security* (2020).

[46] A. S. Khader and D. Lai. ‘Preventing man-in-the-middle attack in Diffie-Hellman key exchange protocol’. In: *2015 22nd international conference on telecommunications (ICT)*. IEEE. 2015, pp. 204–208.

[47] S. Nikoletseas and J. D. Rolim. *Theoretical aspects of distributed computing in sensor networks*. Springer, 2011.

[48] N. P. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Springer, 2016.

[49] M. J. Dworkin. ‘SHA-3 standard: Permutation-based hash and extendable-output functions’. In: (2015).

[50] I.-C. Yeh and C.-h. Lien. ‘The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients’. In: *Expert Systems with Applications* 36.2 (2009), pp. 2473–2480.

[51] S. Moro, P. Cortez and P. Rita. ‘A data-driven approach to predict the success of bank telemarketing’. In: *Decision Support Systems* 62 (2014), pp. 22–31.

[52] M. Antonini, M. Barlaud, P. Mathieu and I. Daubechies. ‘Image coding using wavelet transform’. In: *IEEE Transactions on image processing* 1.2 (1992), pp. 205–220.

[53] J. Patel and C. Read. *Handbook of the Normal Distribution, Second Edition*. Statistics: A Series of Textbooks and Monographs. Taylor & Francis, 1996. ISBN: 9780824793425. URL: https://books.google.nl/books?id=ey61Zm0f0qoC.

[54] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu and W. Shi. ‘On security analysis of proof-of-elapsed-time (poet)’. In: *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer. 2017, pp. 282–297.

[55] A. Corso. ‘Performance analysis of proof-of-elapsed-time (poet) consensus in the sawtooth blockchain framework’. PhD thesis. University of Oregon, 2019.

# 4

# Taming Malicious Majority of Clients in Federated Learning

*Byzantine-robust Federated Learning (FL) aims to counter malicious clients and train an accurate global model while maintaining an extremely low attack success rate. Most of the existing systems, however, are only robust when most of the clients are honest.* `FLTrust` *(NDSS '21) and* `Zeno++` *(ICML '20) do not make the honest majority assumption but can only be applied to scenarios where the server is provided with an auxiliary dataset before training to filter malicious updates.* `FLAME` *(USENIX '22) and* `EIFFeL` *(CCS '22) maintain the semi-honest majority assumption but instead guarantee both robustness and updates confidentiality. It is therefore currently impossible to ensure Byzantine robustness without assuming a semi-honest majority and provide updates confidentiality. To tackle this problem, we propose a novel Byzantine-robust and privacy-preserving FL system, called* `MUDGUARD`*, to capture malicious minority and majority for server and client sides. Specifically, based on DBSCAN, we design a new method via extracting features of updates by pairwise adjusted cosine similarity to boost the clustering accuracy. To thwart attacks of a malicious majority, we develop a method called Model Segmentation, where local updates in the same cluster are aggregated together, and the aggregations are sent back to corresponding clients correctly. We also leverage multiple cryptographic tools to conduct clustering tasks without sacrificing training correctness and updates confidentiality. We present detailed security proof and empirical evaluation along with a convergence analysis for* `MUDGUARD`*. Our experimental results demonstrate that the accuracy of* `MUDGUARD` *is practically close to the FL baseline using FedAvg without attacks (≈0.8% gap on average). Meanwhile, the attack success rate is around 0%-5% even under adaptive attack tailored to* `MUDGUARD`*. We further optimize our design by binary secret sharing and polynomial transformation so that the communication overhead and runtime can be decreased by 67%-89.17% and 66.05%-68.75%, respectively.*

## 4.1. Introduction

Thanks to its privacy properties, Federated Learning (FL) [1] has been widely applied in real-world applications, e.g., prediction of the future oxygen requirements of symptomatic patients with COVID-19 [2]. Despite its attractive benefits, FL is vulnerable to Byzantine attacks. For example, attackers may choose to deteriorate the testing accuracy of models in an untargeted attack. Alternatively, they might fool models to predict an attack-chosen label without downgrading the testing accuracy in a targeted attack. Many research works [3–5] have proved the vulnerability of FL via well-designed attack methods, e.g., poisoning training data or manipulating updates. Other studies [6–13] have been dedicated to strengthening FL assuming that a minority of the clients can be malicious and that the server is honest.

Beyond Byzantine attacks, FL could put clients at high risk of privacy breach [14, 15] even if clients' datasets are maintained locally. Several studies [10, 12, 16] have applied secure tools, e.g., Additive Homomorphic Encryption (AHE) [17], Differential Privacy (DP) [18, 19], and Secure Multiparty Computation (MPC), to protect clients' updates[1]. However, these works only guarantee security when all servers are (semi-)honest and when a minority of the clients are malicious.

To the best of our knowledge, there does not exist any FL system that is capable of withstanding the presence of a majority of Byzantine clients, as well as malicious servers, while also guaranteeing the confidentiality of updates. One may think that existing Byzantine-robust solutions could be trivially extended to address the above challenge. However, that is not the case because they either violate privacy preservation requirements or are only effective in the honest majority scenario. For example, `FLTrust` [9] and `Zeno++` [11] require an auxiliary dataset that is independently and identically distributed (iid) with the clients' training datasets to rectify malicious updates, which evidently violates the clients' privacy. As for `FLAME` [10], it clusters updates and considers the smallest cluster as a malicious group, which makes sense in the malicious minority context. However, in the case of a malicious majority, it is difficult to assert if a given large/small-size cluster is malicious. `EIFFeL` [12] shows similar infeasibility, since it combines existing Byzantine-robust methods (e.g., `FLTrust` [9] and `Zeno++` [11]) with secure aggregation [20].

**Contributions.** We propose a practical and secure Byzantine-robust FL system, `MUDGUARD`, that defends against malicious entities (i.e., malicious minority for servers and malicious majority for clients) with privacy preservation. Specifically, we perform feature extraction on the client updates by calculating the pairwise adjusted cosine similarity. These extracted features are taken to the DBSCAN clustering, which calculates the pairwise $L_2$ distance between the inputs and determines clusters based on density. Subsequently, a new method, *Model Segmentation*, aggregates the updates based on their assigned cluster labels. The aggregated results are returned to clients in the respective clusters. Moreover, the integration of cryptographic tools with the aforementioned calculations allows for the establishment of both Byzantine robustness and privacy preservation within the `MUDGUARD` system. In spite of the

[1]Note AHE and MPC require onerous computation over ciphertexts so that the computational complexity could naturally increase.

| Aggregation strategy | Threat model | | Byzantine robustness | Updates confidentiality | No requirement for an auxiliary dataset | Computation complexity | Communication complexity |
|---|---|---|---|---|---|---|---|
| | Malicious server(s) | Malicious majority clients | | | | | |
| Zeno++ [11] | ✗ | ✓ | ✓ | ✗ | ✗ | $O(d)$ | $O(nd)$ |
| FLTrust [9] | ✗ | ✓ | ✓ | ✗ | ✗ | $O(n)$ | $O(nd)$ |
| FLAME [10] | ✗ | ✗ | ✓ | ✓ | ✓ | $O(d(S^2+n^2))$ | $O(d^2+Sn^2)$ |
| EIFFeL [12] | ✗[1] | ✗ | ✓ | ✓ | ✓ | $O((n+d)n\log^2 n\log\log n+md\min(n,m^2))$ | $O(n^2+md\min(n,m^2))$ |
| MUDGUARD(Ours) | ✓ | ✓ | ✓ | ✓ | ✓ | $O(d+n^3)$ | $O(S(d+n^2))$ |

*d* stands for the dimension of a model. $n, m$, and $S$ represent the number of clients, malicious clients, and servers, respectively.

[1] EIFFeL considers a malicious server to be one that infers privacy information from other parties, which is equivalent to a semi-honest server in our context.

Table 4.1: Comparison of FL systems

utilization of clustering for Byzantine robustness in [10], its approach relies on the assumption that only less than half of clients are malicious, since updates are directly incorporated into HDBSCAN clustering. In contrast to [10], our proposed methodology involves preliminary feature extraction of updates before sending them to the clustering algorithm and separate aggregation, accommodating non-iid and malicious majority clients. We stress that [10] is vulnerable to the non-iid and majority of malicious clients problems due to the absence of pre-processing of updates and cluster detection. To the best of our knowledge, MUDGUARD is the first Byzantine-robust FL system that is able to defend against malicious majority clients without sacrificing clients' privacy. We also are the first to provide detailed security and privacy analysis under Universal Composability (UC) in FL. In the literature on centralized learning, a few MPC-based solutions [21–23] were proposed for model training and malicious servers under UC. Note that the key differences between these works and ours are the required computing functionalities for clustering and model training, as well as the required security properties of the solution. The impact of these works on performance and security is unknown when applied to FL.

We summarize the advantages of MUDGUARD on the SOTA FL systems in Table 4.1. For a theoretical and empirical analysis of complexity, please refer to Section 4.4.7 and Section 4.5.2. Our main contributions can be described as follows.

• We formulate a new aggregation strategy, *Model Segmentation*, for Byzantine-robust FL to effectively avoid poisoning attacks from a majority of malicious clients without requiring the servers to own an auxiliary dataset. It posits that the utilization of complex algorithms for the detection of malicious updates is not necessary. Instead, it only suggests implementing measures to prevent the co-existence of malicious and semi-honest clients in one aggregation.

• We propose a new method to improve the accuracy of updates clustering under non-iid scenarios. Instead of using the updates directly for clustering, we first compute the pairwise adjusted cosine similarity of updates (featured by different directions and magnitudes of updates between every two clients). Then we input the results to DBSCAN.

• We design a secure FL system to be compatible with the cryptographic tools under the malicious context. To protect the updates on the server side and guarantee all clients receive correct aggregations, we construct a secure DBSCAN clustering that leverages cryptographic tools and secure aggregation with Homomorphic Hash

Function (HHF) [24]. We further optimize the secure computations on the server side based on binary secret sharing and polynomial transformation.

• We provide a formal security proof for `MUDGUARD` in the UC framework. This proof captures dynamic security requirements, making `MUDGUARD` more practical than theoretical in security. `MUDGUARD` is the first UC-secure type in the research line of privacy-preserving FL.

• We implement `MUDGUARD` and perform evaluations on (F)MNIST and CIFAR-10 to quantify its accuracy under untargeted attacks, the Attack Success Rate (ASR) under targeted attacks or under an adaptive attack tailored to `MUDGUARD`, as well as its runtime and communication costs. Our experimental results show that the model trained by `MUDGUARD` maintains comparable testing accuracy with the FL baseline - a "no-attack-and-protection" FL with only honest parties (≈0.8% gap on average under untargeted attacks). The ASR under the targeted attacks is as low as 0%-5%. After optimizing the cryptographic computations, the runtime and communication costs are reduced by about 66.05%-68.75% and 67%-89.17%, respectively. For example, in the training of ResNet-18 using CIFAR-10, our optimization strategy can reduce training time from 95 seconds to 48 seconds and communication costs from 16331 MB to 5909 MB, whereas a vanilla FL takes nearly 24 seconds and 758 MB per round.

## 4.2. Background and Related Work

### 4.2.1. Attacks against Federated Learning

**Byzantine Attacks.** Malicious clients may attempt to deteriorate the testing accuracy of the global model by intentionally uploading poisoned updates (i.e., untargeted attacks). Instead of harming the accuracy, the attackers may also intentionally use samples with triggers to launch attacks that make the model misclassify (i.e., targeted attacks). In the following, we review some classical and SOTA untargeted attacks (Gaussian Attack [3], Label Flipping Attack [25], Krum Attack and Trim Attack [3]) and targeted attacks (Backdoor Attack [26] and Edge-case Attack [27]).

• **Gaussian Attack (GA).** Malicious clients degrade the model accuracy by uploading local updates randomly sampled from a Gaussian distribution.

• **Label Flipping Attack (LFA).** Malicious clients flip the local data labels to generate faulty gradients. In particular, the label of each sample is flipped from $y$ to $L-1-y, y \in [L]$, where $L$ is the total number of classes.

• **Krum and Trim Attacks.** These two untargeted local model poisoning attacks are optimized for `Krum` [6] and the `Trim-mean/Median` [7] aggregation strategies, respectively. They aim to pull the global model towards the opposite direction of the honest gradient when it is updated. Besides, they also have attack efficacy on `FedAvg`.

• **Backdoor Attack (BA).** Byzantine clients embed triggers to training samples and change their labels to targeted labels. Their goal is to make the global model misclassify the correct labels to the targeted ones when testing samples with triggers.

• **Edge-case Attack (EA).** The attack aims to misclassify seemingly similar inputs that are unlikely to be part of the training or testing data. For example, by labeling

Ardis[2] 7 images as 1 and adding them to training data, EA can easily backdoor an MNIST classifier. Similarly, the attack can use a Southwest airplanes dataset labeled truck to inject a backdoor into a CIFAR-10 classifier. Note that the attack relies on a restricted assumption that an extra dataset resemblance to the training dataset should be given.

To defend against these attacks, we propose a new approach called *Model Segmentation* in conjunction with feature-extracted DBSCAN. Different from other Byzantine-robust FL, our proposed method generates multiple global models and does not require servers to detect whether a particular group is benign or not. It aggregates only updates with the same cluster labels and returns the aggregations to the corresponding clients. This ensures that updates with similar directions and magnitudes are aggregated together (i.e., benign updates are not aggregated with malicious updates), providing a guarantee of Byzantine robustness in the case of malicious majority clients. Different from `FLAME`, `MUDGUARD` first uses pairwise adjusted cosine similarity to perform feature extraction on updates, then clusters through DBSCAN. The advantage of this is that it can reduce the false positive rate and be effective in non-iid situations. For detailed explanations, please refer to Section 4.4.2 and Section 4.6. The experimental results show that the Byzantine robustness and clustering accuracy of `MUDGUARD` is better than that of `FLAME` as will be shown later in Section 4.5.1.

**Inference Attacks.** Although local datasets are not directly revealed during the FL training process, the updates are still subject to privacy leakage if the server is semi-honest or even malicious [14, 28, 29]. For instance, Zhu *et al.* [14] investigated a method of training data reconstruction via optimizing the $L_2$ distance between uploaded gradients and gradients trained from dummy samples using an L-BFGS solver. This approach allows servers to easily reconstruct the local datasets and achieves even pixel-wise accuracy for images and token-wise matching accuracy for texts. To defend against such attacks, we use Secret Sharing (SS) [30] to split client updates into $s$ shares before sending them to servers. This way, updates are safeguarded from malicious servers since they do not get sufficient shares to perform update reconstruction. Even if malicious servers collude with malicious clients, no extra benefit can be achieved towards compromising the update shares belonging to semi-honest clients.

**Differential Attack.** We use DBSCAN in conjunction with *Model Segmentation* to separate benign and malicious updates. Features extraction with adjusted cosine similarity greatly descends the likelihood of false positives. However, we cannot guarantee that the clustering results are 100% correct. A semi-honest client could be clustered together with other malicious clients by a small probability as it could have a smaller similarity from semi-honest clients than that from malicious clients. In particular, this case happens more frequently in SignSGD, where only taking signs of gradients to update the model because the algorithm computing adjusted cosine similarity disregards the magnitude of the gradients, resulting in the same effect as calculating cosine similarity. The above phenomenon triggers the differential

[2]A dataset extracted from 15,000 Swedish church records written by different priests with various handwriting styles in the nineteenth and twentieth centuries.

attack in the following cases. Assuming that, at $t$-th round, a semi-honest client is misclustered to a malicious group. After returning aggregation to the group, the benign updates can be easily revealed by subtracting those malicious updates, and the inference attack can be launched further. Another more common case is that a malicious adversary $\mathscr{A}$ compromises $m$ clients and then makes one of them perform correct operations, i.e., acting as a semi-honest client. This malicious-but-act-semi-honest client, being assigned to a semi-honest group, can get benign aggregation from each round and then conduct an inference attack. In this work, we apply DP to make aggregations obtained by malicious clients statistically indistinguishable from those containing benign updates, guaranteeing that benign updates cannot be easily identified from aggregations.

### 4.2.2. Defenses

**Byzantine-robust Federated Learning.** Blanchard *et al.* [6] proposed `Krum` to select 1 out of $n$ (local updates) as a global update for each round, where the selected updates should have the smallest $L_2$ distance from others. Yin *et al.* [7] introduced `Trim-mean` and `Median` to resist Byzantine attacks. The former uses a coordinate-wise aggregation strategy. The server calculates $n-2z$ values for each model parameter as the global model update, wherein the largest and smallest $z$ values are filtered. Unlike `FedAvg` [1] computing the weighted average of all parameters, the latter calculates the median of parameters. This median serves as an update to the global model. A major drawback of the aforementioned mechanism is that it is effective only under a majority of honest clients working with an honest server. In `Median` [7], the median calculated by the server can easily be malicious if malicious clients control a large/overwhelming proportion of updates. This similarly applies to `Trim-mean` and `Krum`. Cao *et al.* [9] proposed `FLTrust` to protect against a malicious majority at the client side, assuming an honest server holds a small auxiliary dataset. The server treats the gradients trained from this small dataset as the root of trust. By comparing these trusted results with the updates sent by clients, the server can easily rule out malicious updates. Under the same assumption, `Zeno++` [11] uses an auxiliary dataset to calculate the loss value of each local model. A client is determined to be honest if the loss value is beyond the preset threshold. While using an auxiliary dataset could be intriguing, such approaches are not feasible in the context of FL as they violate the fundamental premise of FL in which local datasets are not to be shared with any parties.

**Privacy-preserving Federated Learning.** Truex *et al.* [16] proposed a solution enabling clients to use AHE and DP to secure gradients in the semi-honest context (for both clients and the server). Since DP noise is applied on gradients, the accuracy of the global model is deteriorated. In the scenario of honest majority clients with two semi-honest servers, Thien *et al.* [10] proposed `FLAME` using an MPC protocol to protect gradients from the servers and enabling the servers to perform clustering for Byzantine robustness. Specifically, the clients can securely share their updates to the servers cryptographically, e.g., via secret sharing, and the servers can filter out malicious updates without knowing their concrete values. By expressing existing Byzantine-robust solutions (e.g., `FLTrust`) as arithmetic circuits, `EIFFeL` [12] enables secure aggregation of verified updates. Although FLAME and EIFFeL capture both Byzantine robustness and privacy preservation (i.e., update confidentiality), the accuracy of the global model could become equivalent to a random guess if the proportion of malicious clients is ≥50%.

## 4.2.3. Tools

### Federated Learning

Federated Learning (FL) enables $n$ clients to train a global model $\boldsymbol{w}$ collaboratively without revealing local datasets. Unlike centralized learning, FL requires clients to upload the weights of local models ($\{\boldsymbol{w}^i \mid i \in n\}$) to a parametric server. It aims to optimize a loss function: $\arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} \frac{k_i}{K} \mathscr{L}_i(\boldsymbol{w}, \mathscr{D}_i)$, where $\mathscr{L}_i(\cdot)$ and $k_i$ are the loss function and local data size of $i$-th client. At $t$-th round, the training of FL can usually be divided into the following steps.

• *Global model download*: The server selects partial clients engaging in training. All connected clients download the global model $\boldsymbol{w}_{t-1}$ from the server.

• *Local training*: Each client updates its local model by training with its own dataset: $\boldsymbol{g}^i_{t-1} \leftarrow \frac{\partial \mathscr{L}(\boldsymbol{w}_{t-1}, \mathscr{D}_i)}{\partial \boldsymbol{w}_{t-1}}$.

• *Aggregation*: After the local updates $\{\boldsymbol{g}^i_{t-1} \mid i \in n\}$ are uploaded, the server updates the global model by aggregation: $\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1} - \eta \sum_{i=1}^{n} \frac{k_i}{K} \boldsymbol{g}^i_{t-1}$, where $\eta$ refers to the learning rate.

### DBSCAN

Unlike traditional clustering algorithms (e.g., k-means, k-means++, bi-kmeans), which need to pre-define the number of clusters, Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [31] is proposed to cluster data points dynamically. Based on density-based clustering, DBSCAN guarantees that clusters of any shape can always be identified. Besides, it can recognize noise points effectively. Basically, after setting the density parameter ($\alpha$) and the minimum cluster size ($mPts$), DBSCAN can conduct effective clustering. We note HDBSCAN [32] could be also used for clustering. Its main difference from DBSCAN is the multiple densities clustering. In this work, we assume that malicious clients may only conduct one kind of attack during the whole training, e.g., a group of malicious clients conducting a Label Flipping Attack together. The malicious updates could only derive one density. Like [10], we may apply HDBSCAN in the clustering. But DBSCAN, in general, requires less computational complexity than HDBSCAN in algorithmic constructions. And further, we will conduct the clustering with cryptographic operations. Considering efficiency, we choose DBSCAN over HDBSCAN.

### Cryptographic Tools

The secure Multiparty Computation (MPC) framework aims to enable multiple parties to evaluate a function over ciphertexts securely. The parties conducting MPC can access inputs via protection approaches, e.g., in a secret-shared format. It does not leak any information besides the final output unless these shares are combined to derive plaintexts.

**Secret Sharing (SS).** It refers to a type of tool for splitting a secret among multiple parties, each of whom is assigned a share of the secret. The security of an SS scheme guarantees that one can distinguish shares and randoms with a negligible probability. Apart from that, no one can reconstruct the secret unless holding all (or a subset) of shares. Let us consider Shamir Secret Sharing (SSS) [30] $(t, n)$-threshold scheme as an example. Assume one chooses a polynomial $f(x) = \sum_{i=0}^{t-1} a_i \cdot x^i$ over $\mathbb{Z}_q$ and a secret $a_0 = f(0)$. The secret can be split into $n$ shares by randomly selecting $n$ values: $\{r_j \leftarrow \mathbb{Z}_q^* \mid j \in n\}$, and then calculating shares $\{f(r_j) \mid j \in n\}$. Given a subset of any $t$ out of $n$ shares, the secret can by reconstructed by Lagrange interpolation [33]:$f(0) = \sum_{j=0}^{t-1} f(r_j) \cdot \prod_{z=0, z\neq j}^{t-1} \frac{r_z}{r_z - r_j}$. Except for SSS, other schemes like additive SS and replicated SS are used in the MPC framework [34, 35]. Note these schemes have a linear property. Even if each party performs linear combinations locally with shares, the combined secret matches the result obtained by these linear calculations. This saves significant communication costs in the FL context, where servers are only required to aggregate shares of gradients.

**Homomorphic Hash Functions (HHF).** Given a message $x \in \mathbb{Z}_q$, a collision-resistant HHF [24] $\mathsf{H}$: $\mathbb{G}_1 \times \mathbb{G}_2 \leftarrow \mathbb{Z}_q$ can be indicated as $\mathsf{H}(x) = (g^{\mathsf{H}'_{\delta,\phi}(x)}, h^{\mathsf{H}'_{\delta,\phi}(x)})$, where $\delta$ and $\phi$ are secret keys randomly and independently selected from $\mathbb{Z}_q$. $\mathsf{H}'$ is a hash function, and $\mathbb{G}_1$ and $\mathbb{G}_2$ are two different groups. Similar to other one-way hash functions, the security of the HHF requires that one can find a collision with a negligible probability. Based on additive homomorphism: $\mathsf{H}(x_1 + x_2) \leftarrow (g^{\mathsf{H}'_{\delta,\phi}(x_1)+\mathsf{H}'_{\delta,\phi}(x_2)}, h^{\mathsf{H}'_{\delta,\phi}(x_1)+\mathsf{H}'_{\delta,\phi}(x_2)})$, in this work, we will use this tool as a verification of the correctness of aggregation.

**Homomorphic Encryption (HE).** This tool is an interesting privacy-preserving technology enabling users to evaluate polynomial computations on ciphertexts without revealing underlying plaintexts. An encryption scheme is called partial HE if it only supports addition [17] or multiplication [36], while fully HE [37] can support both. An HE scheme usually includes the following steps.

- *Key Generation:* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}\ (1^\lambda)$, where based on security level parameter $\lambda$, public key $\mathsf{pk}$ and secret key $\mathsf{sk}$ can be generated.
- *Encryption:* $(c_1, c_2) \leftarrow \mathsf{Enc}(\mathsf{pk}, m_1, m_2)$. By using $\mathsf{pk}$, the probabilistic algorithm encrypts messages $m_1, m_2$ to ciphertexts $c_1, c_2$.
- *Homomorphic evaluation:* $\mathsf{Eval}(c_1, c_2) = c_1 \circ c_2 = \mathsf{Enc}(\mathsf{pk}, m_1) \circ \mathsf{Enc}(\mathsf{pk}, m_2) = \mathsf{Enc}(\mathsf{pk}, m_1 \circ m_2)$, where $\circ$ refers to an operator, e.g., addition or multiplication.
- *Decryption:* $m_1 \circ m_2 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m_1 \circ m_2))$. Using $\mathsf{sk}$, the operational results of $m_1$ and $m_2$ can be derived.

**Oblivious Transfer (OT).** OT [38] is one of the crucial building blocks for MPC. In an OT protocol (involving two parties), a sender holds $n$ different strings $s_i, i = 1 \cdots n$, and a receiver has an index $(ind)$ and wants to learn $s_{ind}$. At the end of the protocol, the receiver cannot get information about strings rather than $s_{ind}$, while the sender learns nothing about $ind$ selected by the receiver. For example, a 1-out-of-2 OT protocol only inputs two strings and a 1-bit index.

**Garbled Circuits (GC) [39].** The protocol is run between two parties called the garbler and evaluator. The garbler generates the GC corresponding to the Boolean circuit to be evaluated securely by associating two random keys per wire representing the bit values 0, 1. The garbler then sends the GC together with the keys for the inputs to the evaluator. The evaluator obliviously obtains the keys for his inputs via OT and evaluates the circuit to obtain the output key. Finally, the evaluator maps the output key to the real output.

### DIFFERENTIAL PRIVACY

Differential Privacy (DP) [18] is a data protection approach enabling one to publish statistical information of datasets while keeping individual data private. The security of DP requires that adversaries cannot statistically distinguish the changes between two datasets where an arbitrary data point is different. The most widely used DP mechanism is called $(\epsilon,\delta)$-DP defined below, requiring less injection noise than the $\epsilon$-DP but standing at the same privacy level.

**Definition 4.2.1** ($(\epsilon,\delta)$ - Differential Privacy [18])**.** *Given two real positive numbers* $(\epsilon,\delta)$ *and a randomized algorithm* $\mathscr{A}$: $\mathscr{D}^n \rightarrow \mathscr{Y}$, *the algorithm* $\mathscr{A}$ *provides* $(\epsilon,\delta)$ *- DP if for all data sets* $\mathbf{D},\mathbf{D}^{'} \in \mathscr{D}^n$ *differing in only one data sample, and all* $\mathscr{S} \subseteq \mathscr{Y}$*:* $Pr[\mathscr{A}(\mathbf{D}) \in \mathscr{S}] \leq exp(\epsilon) \cdot Pr[\mathscr{A}(\mathbf{D}^{'}) \in \mathscr{S}] + \delta$.

Note that the Gaussian noise $\mathscr{N} \sim N(0,\Delta^2\sigma^2)$ should be added to the output of the algorithm, where $\Delta$ is $L_2$ sensitivity of $\mathbf{D}$ and $\sigma=\sqrt{2\ln(1.25/\delta)}$ [40]. The robustness of post-processing guarantees for any probabilistic/deterministic functions $\mathscr{F}$, if $\mathscr{A}$ satisfies $(\epsilon,\delta)$-DP, so does $\mathscr{F}(\mathscr{A})$.

## 4.3. PROBLEM FORMULATION

### 4.3.1. SYSTEM MODEL

Before proceeding, we provide some assumptions about MUDGUARD. We assume training is conducted on a dataset $\mathscr{D}$ with $K$ data samples composed with feature space $\mathscr{X}$ (each sample containing all features) and a label set $\mathscr{Y}$. Additionally, $\mathscr{D}$ is horizontally partitioned among $n$ clients, indicated as $\mathscr{X}_i = \mathscr{X}_j, \mathscr{Y}_i = \mathscr{Y}_j, \mathscr{I}_i \cap \mathscr{I}_j = \emptyset, \forall \mathscr{D}_i, \mathscr{D}_j, i \neq j$, where all clients share the same feature space and labels but differ in sample index space $\mathscr{I}$. FL aims to optimize a loss function: $\operatorname*{argmin}_{\boldsymbol{w}} \sum_{i=1}^{n} \frac{k_i}{K} \mathscr{L}_i(\boldsymbol{w},\mathscr{D}_i)$, where $\mathscr{L}_i(\cdot)$ and $k_i$ are the loss function and local data size of $i$-th client.

For reasons that relate to the versatility of the FL system, we also consider $S$ $(>2)$ servers to carry out clustering and aggregation (e.g., FedAvg). This allows us to protect from malicious servers who cannot reconstruct the secrets so long as their number is less than $S/2$ by using cryptographic tools, in which clients send updates in secret-shared format. We state that our Byzantine solution can also be executed by only one server. In this case, considering privacy, we have to assume that the server must be fully trusted or semi-honest. Note that our focus here is on the existence of malicious servers. In this research line [21, 41], secure computation is considered among multiple servers. Due to page limit, we summarize frequently used notations in Table 4.2.

### 4.3.2. Threat Model

We mainly consider potential threats incurred by participating clients, servers, and outside adversaries.

• **Attackers' goal.** We assume that two different entities are involved in the training: semi-honest and dynamic malicious parties (including servers and clients), in which both try to infer the privacy (updates) information of others from the received messages. Unlike the former, strictly following the designed algorithms, the malicious clients additionally aim to deteriorate the performance or boost the ASR of the global model through untargeted or targeted poisoning attacks, respectively.

• **Attackers' capabilities.** The malicious servers (in a minority proportion) and clients (in a majority proportion) can deviate from the designed protocols. For example, the malicious servers can perform an incorrect aggregation and send it back to the semi-honest group. Moreover, malicious parties (servers and clients) can collude with each other to infer benign aggregations and maximize the efficacy of poisoning attacks (e.g., the Krum attack). To resist outside adversaries, secret-shared messages are transmitted by private communication channels. Other messages are transmitted through public communication channels, where outsiders are allowed to eavesdrop on these channels and try to infer clients' (updates) privacy during the whole training phase.

• **Attackers' knowledge.** We assume that the loss function, data distributions, Byzantine-robust aggregation strategy, and public parameters (including training and security parameters) are revealed to all parties. The malicious clients can exploit this information to design and cast adaptive attacks tailored to MUDGUARD. For privacy reasons, the local updates and datasets of semi-honest clients are not revealed to malicious parties.

## 4.4. MUDGUARD Overview and Design

### 4.4.1. Overview

In a traditional FL system, clients send updates to the servers for global model aggregation. Considering there exist malicious clients, we should maintain the Byzantine robustness such that malicious updates should be excluded properly. To do so, the servers must separate the malicious clients from the semi-honest clients. DBSCAN helps the servers to perform clustering. Since the main difference between the malicious and the benign is in the direction and magnitude of the updates, we use the adjusted cosine similarity of updates as feature extraction to obtain better clustering accuracy. Under the (semi-)honest majority, the clustering result directly links to the group size. However, for a dynamic malicious majority, we cannot judge if a cluster is malicious only based on its size. To address this issue, we propose *Model Segmentation*. Unlike traditional FL generating "a unique" global model, our proposed algorithm can yield multiple aggregation results. It does not

| Notation | Description |
|---|---|
| $\boldsymbol{g}_t^i$ | gradients of $i$-th client at $t$-th round |
| $\boldsymbol{w}_t^i$ | weights of $i$-th client at $t$-th round |
| $T$ | the number of rounds |
| $n$ | the number of clients |
| $S$ | the number of servers |
| $m$ | the number of malicious clients |
| $k_i$ | the number of data instances of $i$-th client |
| $c$ | the number of clusters |
| $l$ | the cluster labels |
| $E$ | the number of epochs |
| $\mathscr{D}_i$ | the dataset of $i$-th client |
| $\eta$ | learning rate |
| $\boldsymbol{G}^z$ | the aggregation of gradients of $z$-th cluster |
| $[n]$ | a set of numbers ranging from 1 to n |
| $[\![\cdot]\!]$ | secret shared format |
| CosM | pairwise adjusted cosine similarity matrix |
| EudM | pairwise $L_2$ distance matrix |
| IndM | indicator matrix |
| $\delta, \phi$ | secret keys of homomorphic hash function |
| $\Delta, \sigma, \epsilon$ | parameters of differential privacy |
| $\mathcal{N}$ | Gaussian noise |
| $\alpha$ | density parameter |
| ECD$(\cdot)$ | encoding algorithm |
| DCD$(\cdot)$ | decoding algorithm |

Table 4.2: Notation summary

require the servers to know whether a given group is malicious or not. Moreover, it only aggregates the updates within the same cluster and then returns the results to the corresponding clients. We thus guarantee that the semi-honest will not be aggregated with the malicious.

As far as fighting against inference attacks is concerned, we should protect the confidentiality of the updates. For this, we use SS to wrap the updates into a secret shared format in the sense that individual secret shares cannot reveal the underlying information of the updates. By doing so, we guarantee that the updates are secured from eavesdroppers, semi-honest, or even malicious servers and further can be used on secure multiplication, comparison, and aggregation via cryptographic tools. However, using SS alone is not sufficient to defend against differential attacks. To thwart the attack, we apply DP to prevent the attackers from extracting benign updates from the semi-honest group. Since injecting noise brings a negative influence on the accuracy of the training model, we enable clients to perform denoising before wrapping the results into shares. Note that this does not invalidate DP due to the post-processing nature [18]. We also consider the malicious minority servers and thus leverage HHF to prevent malicious servers from performing incorrect aggregation, e.g., merging the gradients from two different groups.

## 4.4.2. BYZANTINE-ROBUST AGGREGATION STRATEGY WITH CRYPTOGRAPHIC COMPUTATIONS

Our workflow of the Byzantine-robust aggregation strategy is as follows. Firstly, the clients upload the gradients of the local models to the server side. Secondly, servers extract features of gradients and split gradients into multiple clusters via DBSCAN. Finally, servers aggregate the gradients in the clusters separately and send aggregations to the corresponding clients. In the following, we complete the strategy over secure cryptographic computations.

**Gradients Upload.** The use of the pairwise adjusted cosine similarity matrix (CosM) as a method for extracting features is motivated by the fact that it measures both the difference in directions and magnitudes of updates. This is particularly useful when dealing with clients exhibiting various behaviors and non-iid cases. In this context, CosM and $L_2$ distance are used as input and the metric of DBSCAN, respectively. The most direct method of computing CosM is as follows. We first subtract updates with their mean values. For the updates of each client, we compute the pairwise dot product and $L_2$ norm to derive the numerator and denominator, respectively, and then we can calculate CosM from the division (of numerator and denominator). The above operations become inefficient if the processing is carried out using cryptographic tools. The servers are required to perform the computations of shared mean, numerator, denomination, and then division to finally get the shared adjusted cosine similarity matrix $[\![\mathsf{CosM}]\!]$.

To improve efficiency and optimize the above method, we consider the denoised gradients of client $i$ at $t$-th round $\hat{\mathbf{g}}_t^i$ as updates and perform binary secret sharing via SignSGD. Note that SignSGD only takes the signs of gradients to the update model, resulting in benign and malicious having the same magnitudes. Thus, in this case, we can easily compute the adjusted cosine similarity via simple bit-wise XORing. Figure 4.1 depicts this optimization procedure.

Therefore, in each training round (of the optimization), client $i$ derives the gradients using SGD [42]. Considering the upcoming cryptographic clustering, one needs to compute the signs of gradients $\mathsf{sign}(\hat{\mathbf{g}}_t^i) \in \{-1,+1\}$ as SignSGD [43] and then encodes to Boolean representation, which is compatible with binary SS and XOR operations. Without loss of generality, we implement a widely used encoding/decoding method as

$$\mathsf{ECD}(\mathsf{sign}(\hat{\mathbf{g}}_t^i)) = \begin{cases} 1, & \mathsf{sign}(\hat{\mathbf{g}}_t^i) = +1 \\ 0, & otherwise \end{cases},$$

$$\mathsf{DCD}(\mathsf{ECD}(\mathsf{sign}(\hat{\mathbf{g}}_t^i))) = 2\mathsf{ECD}(\mathsf{sign}(\hat{\mathbf{g}}_t^i)) - 1.$$

This method guarantees $\mathsf{DCD}(\mathsf{ECD}(\mathsf{sign}(\hat{\mathbf{g}}_t^i))) = \mathsf{sign}(\hat{\mathbf{g}}_t^i)$. Each client sends the encoded updates to the servers via binary SS and broadcasts the hash results of unencoded updates for future verification. Although SignSGD is lightweight, it brings a negative impact on the accuracy of clustering. Section 4.5.1 provides a detailed analysis of this impact. Note that SignSGD has the natural capability of defending against scaling attacks [26] since it only takes signs as updates and clips

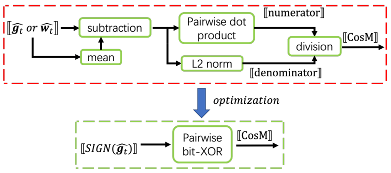


Figure 4.1: Optimization on the calculation of adjusted cosine similarity. - - - - : optimized process: by binary SS, [[CosM]] is computed via bit-XOR. : unoptimized process: taking gradients or weights as updates to compute [[CosM]].

the magnitude of gradients. An attacker still can easily deteriorate the global model by constructing updates in the opposite direction of benign updates.

**Clustering.** As a crucial variable in FL, updates determine the directions and magnitudes of updating in the model, while Byzantine attackers introduce abnormal updates. Traditional clustering approaches directly use updates as inputs, and cosine similarity as metric [10], causing informative redundancy and blurring obvious features, especially in deep models (e.g., ResNet [44]), thereby producing frequent false positives and negatives. Since the adjusted cosine similarity measures the difference in directions and magnitudes of updates at the same time, we use the pairwise adjusted cosine similarity CosM as a method for extracting features, i.e., CosM and $L_2$ distance, used as the input and the main metric of DBSCAN, respectively. We find that this method is effective in distinguishing the updates because 1) calculating CosM (feature extraction) is equivalent to reducing the informative redundancy of updates to improve the clustering accuracy. 2) Using it to calculate the pairwise $L_2$ distance can expand the pairwise differences which are not clearly computed by CosM. Thus there is a more clear density difference between honest and malicious updates. 3) By subtracting the mean updates, CosM helps to account for reducing the influence of non-iid, allowing for a more accurate comparison of clients' updates. 4) When the model converges, using cosine similarity is inappropriate because even semi-honest clients have updates in different directions. For example, under a Gaussian Attack (GA), the malicious and semi-honest clients become indistinguishable. The accuracy of the global model drops sharply to the level of the initial training. We provide concrete examples in Section 4.6 to demonstrate the advantages of using CosM. Note that this advantage is more notable when the cryptographic tools are not optimized. Since the proposed optimization uses SignSGD to align the magnitudes of updates, computing cosine similarity on it naturally provides the same effect on clustering as adjusted cosine similarity.

Next, we describe the process of our clustering. We first extract features (different updates directions with magnitudes) - calculating the CosM $\leftarrow \bar{\mathbf{g}}_t^i \oplus \bar{\mathbf{g}}_t^j, i, j \in [n]$, and then use it as the input for clustering, thereby reducing the rate of false positives. Commonly, the adjusted cosine similarity of two vectors is obtained by first calculating the dot product of the vectors and then dividing them by the product of their respective $L_2$ norm. Since encoding updates $\hat{\mathbf{g}}_t^j \in \{-1,+1\}^d$ to $\overline{\mathbf{g}}_t^j \in \{0,+1\}^d$ is inspired by [45], we compute XOR of $\overline{\mathbf{g}}_t^j$ with $d-2p$ bits, which is equivalent to the result of the dot product of $\hat{\mathbf{g}}_t^j$, where $p$ is the counted number of set bits.

After that, the servers collaboratively compute pairwise $L_2$ distance matrix $\mathsf{EucM}$ using secure multiplications ($vector_{ij} \leftarrow \mathsf{CosM}_i - \mathsf{CosM}_j, i,j \in [n]$, $x_{ij} \leftarrow vector_{ij} \cdot vector_{ij}$, and $\mathsf{EucM}_{ij} \leftarrow 1 + \frac{x_{ij}-1}{2} - \frac{(x_{ij}-1)^2}{8} + \frac{(x_{ij}-1)^3}{16}$) and compare with density $\alpha$ to derive an indication matrix $\mathsf{IndM}$, where $\mathsf{IndM}=1$ if $\mathsf{EucM} \le \alpha$, otherwise $\mathsf{IndM}=0$. Then by applying the DBSCAN, one can derive cluster labels. Note that the main focus of this paper is not on optimizing DBSCAN, we thus do not describe how to retrieve cluster labels from $\mathsf{IndM}$. We refer interested readers to [31].

We see that $\alpha$ has a crucial influence on clustering accuracy. According to the conclusion of Bhagoji *et al.* [46], benign and malicious updates follow normal distributions. We formally derive its upper bound of selection (see Theorem 4.4.1).

**Theorem 4.4.1** (Density Selection)**.** *Suppose the distribution of benign and malicious updates obeys the normal distribution, setting $\alpha < \sqrt{2}$ guarantees that malicious clients conducting a poisoning attack will not be grouped together with benign clients.*

*Proof.* We denote $a = (a_1, \cdots, a_d)$ and $b = (b_1, \cdots, b_d)$ are two vectors uploaded by malicious clients, where $np$ refers to the number of model parameters:

$$Pr(a_i, b_i = \mathsf{sign}) = \begin{cases} \frac{1}{2}, & \mathsf{sign} = +1 \\ \frac{1}{2}, & \mathsf{sign} = -1 \end{cases}, \quad \forall i \in [d].$$

The adjusted cosine similarity can be computed as:

$$\begin{aligned} COS\_similarity &= \frac{a_1 b_1 + \cdots + a_d b_d}{\sqrt{a_1^2 + \cdots + a_d^2} \cdot \sqrt{b_1^2 + \cdots + b_d^2}} \\ &= \frac{a_1 b_1 + \cdots + a_d b_d}{d}. \end{aligned}$$

Since $a_i$ and $b_i$ are relatively independent, we have:

$$Pr(a_i \cdot b_i = \mathsf{sign}) = \begin{cases} \frac{1}{2}, & \mathsf{sign} = +1 \\ \frac{1}{2}, & \mathsf{sign} = -1 \end{cases}, \quad \forall i \in [d].$$

According to the *Law of large numbers*, $E(COS\_similarity) \sim E(a_i b_i) = 0$. This conclusion can be generalized to any two malicious clients, and malicious clients have the same distance as a semi-honest client. Therefore, if we calculate the adjusted cosine similarity vector of two malicious clients, there should be only two elements of difference. The $L_2$ distance of these two vectors is $\sqrt{2}$. □

Note that taking $\alpha < \sqrt{2}$ only allows malicious clients to be identified as noise points, which is not a 100% guarantee that all semi-honest clients are clustered together. Due to the difference in training data, it could happen that the distances between a semi-honest client and other semi-honest clients are greater than $\sqrt{2}$ by chance, resulting in the semi-honest client being identified as a noise point.

**Model Segmentation.** To deal with Byzantine-majority attacks, after obtaining the cluster labels, the servers aggregate the updates within the same cluster and return the results (and their hash values) to the corresponding clients. In our design, unless a malicious client acts honestly, then it will not be grouped into a cluster with the semi-honest clients, with a relatively large probability. This protects benign clients by keeping poisonous updates from global model updates computed for benign clusters. Note here we do not further explore the case in which malicious clients choose to act honestly during training. In fact, if malicious clients behave semi-honestly, we will obtain a more accurate global model. In a sense, this is a bonus for semi-honest clients. After all, in the context of *Model Segmentation*, it is not required to identify malicious groups via any verification algorithms, which is a positive thing since it removes the processing burden from the servers as the latter do not need to run verification over "encrypted-and-noised" updates. Compared with the method of `FLAME`, *Model Segmentation* does not need to assume that most clients in the FL system are (semi-)honest. When integrated with an optimized clustering method, *Model Segmentation* can also enhance the Byzantine robustness of the `MUDGUARD`.

**Resistance against Malicious Servers.** To further prevent malicious servers from casting and sending incorrect aggregation, we use HHF so that every client can verify if the received aggregation is correct. Specifically, the proposed method involves a pre-upload step in which client $i$ broadcasts hash values of signs of gradients $\mathsf{H}_{\delta,\phi}(\mathsf{sign}(\hat{\boldsymbol{g}}t^i))$ to the remaining parties before uploading secret-shared updates to the server side. The servers use additive homomorphism of HHF to calculate hash values of aggregations $\prod_{i\in c_j}\mathsf{H}_{\delta,\phi}(\mathsf{sign}(\hat{\boldsymbol{g}}t^i)) = \mathsf{H}_{\delta,\phi}(\boldsymbol{G}_t^j)$ based on the clustering results, where $c_j$ refers to a cluster $j$ containing client indexes. After receiving the aggregations $\boldsymbol{G}_t^j$, the clients can calculate $\mathsf{H}_{\delta,\phi}(\boldsymbol{G}_t^j)$ and $\prod_{i\in c_j}\mathsf{H}_{\delta,\phi}(\mathsf{sign}(\hat{\boldsymbol{g}}_t^i))$ based on the cluster labels and the received hash values, and subsequently verify whether these two values are equal or not. Note that this method considers the possibility of malicious servers that may send incorrect aggregations and $\mathsf{IndM}$ to the semi-honest clients. However, since only a minority of servers are assumed to be malicious, the semi-honest clients take the most consistent results as the real results.

### 4.4.3. SYSTEM DESIGN

Assume client $i \in [n]$ holds a horizontally partitioned dataset $\mathscr{D}_i$ satisfying $\mathscr{D} = \bigcup_{i=1}^{n} \mathscr{D}_i$, at $t$-th round, `MUDGUARD` works as follows.

**Protocol MUDGUARD**

❶ *Local Training.* For each local minibatch, each client conducts SGD and takes gradients $\boldsymbol{g}_t^i$ as updates.
❷ *Noise Injection.* Each client adds noise into $\boldsymbol{g}_t^i$ to satisfy DP: $\tilde{\boldsymbol{g}}_t^i \leftarrow \boldsymbol{g}_t^i / \max(1, \|\boldsymbol{g}_t^i\|_2/\Delta) + \mathscr{N}(0, \Delta^2\sigma^2)$.
❸ *Denoising.* To improve accuracy, each client denoises $\tilde{\boldsymbol{g}}_t^i$ by $\hat{\boldsymbol{g}}_t^i \leftarrow \mathsf{KS}(\tilde{\boldsymbol{g}}_t^i, \mathscr{N}) \cdot \tilde{\boldsymbol{g}}_t^i$,

where $\mathsf{KS}(\cdot)$ is the KS distance.
❹ *SS.* Each client splits $\overline{\boldsymbol{g}}_t^j \leftarrow \mathsf{ECD}(\mathsf{sign}(\hat{\boldsymbol{g}}_t^j))$ into $S$ shares by binary SS with Tiny Oblivious Transfer (OT) and sends the shares to $S$ servers: $[\![\overline{\boldsymbol{g}}_t^j]\!] \xleftarrow{SS} \overline{\boldsymbol{g}}_t^j$. Besides, by running HHF, all clients broadcast $\mathsf{H}_{\delta,\phi}(\mathsf{sign}(\hat{\boldsymbol{g}}_t^j))$.
❺ *Feature Extraction.* After receiving $n$ shares, each server locally computes a pairwise adjusted cosine similarity matrix by bit-XOR: $[\![\mathsf{CosM}_{ij}]\!] \leftarrow [\![\overline{\boldsymbol{g}}_t^i]\!] \oplus [\![\overline{\boldsymbol{g}}_t^j]\!]$, $i, j \in [n]$. To further compute $L_2$ distance, all servers convert Boolean shares to arithmetic shares by correlated randomness.
❻ *$L_2$ Distance Computation.* After conversion, deriving multiplicative SS, each server uses HE or OT to produce a triple, satisfying further multiplications. Therefore, each server takes $[\![\mathsf{CosM}]\!]$ as the inputs of DBSCAN and then computes $[\![\mathsf{EucM}]\!]$ by (a) pairwise subtraction: $[\![vector_{ij}]\!] \leftarrow [\![\mathsf{CosM}_i]\!] - [\![\mathsf{CosM}_j]\!], i, j \in [n]$, (b) dot product: $[\![x_{ij}]\!] \leftarrow [\![vector_{ij}]\!] \cdot [\![vector_{ij}]\!]$, and (c) approximated square root: $[\![\mathsf{EucM}_{ij}]\!] \leftarrow 1 + \frac{[\![x_{ij}]\!]-1}{2} - \frac{([\![x_{ij}]\!]-1)^2}{8} + \frac{([\![x_{ij}]\!]-1)^3}{16}$.
❼ *Element-wise Comparison.* By comparing each element of $\mathsf{EucM}$ with density parameter $\alpha$, each server can derive shares of indicator matrix $[\![\mathsf{IndM}]\!]$, $\{\mathsf{IndM}_{ij} = 1 \mid \mathsf{EucM}_{ij} \leq \alpha\}$.
❽ *Reconstruction.* All servers run a reconstruction algorithm to reveal $\mathsf{IndM}$: $\mathsf{IndM} \xleftarrow{\mathsf{recon}} [\![\mathsf{IndM}]\!]$ and broadcast it to the client side. By DBSCAN, one can derive cluster labels. Based on these labels, the clients learn about clustering information to perform aggregation verification in step ❿.
❾ *Model Segmentation.* The servers aggregate shares (based on the number of labels $c$) with the same labels after decoding: $\{[\![\boldsymbol{G}_t^j]\!] \leftarrow \sum_{i \in c_j} \mathsf{DCD}([\![\overline{\boldsymbol{g}}_t^i]\!]) \mid c_j = \{i \mid i \in [n]\}, j \in [c],\}$ and send to the corresponding clients.
❿ *Aggregation Verification.* After reconstructing aggregation, according to cluster labels, each client verifies aggregation by $\prod_{i \in c_j} \mathsf{H}_{\delta,\phi}(\mathsf{sign}(\hat{\boldsymbol{g}}_t^i)) \stackrel{?}{=} \mathsf{H}_{\delta,\phi}(\boldsymbol{G}_t^j)$. If the equation holds, clients accept the aggregation results; otherwise, reject and abort.

We note that the corresponding implementation-level algorithms of `MUDGUARD` are given in the following (Algorithms 9 and 10) and will be used in the experiments.

### 4.4.4. Privacy Preservation Guarantee

**Differential attack resistance.** As shown in step ❷ and ❸ of Protocol 4.4.3, each client $i$ can add differentially private noise into gradients and perform denoising later. Like [47], we use KS distance (of noised gradients and noise distribution) as a metric to denoise by multiplying noised gradients. Differentially private updates are first denoised, taken signs, and encoded before being secretly shared.
**Binary SS.** Unlike arithmetic SS in domain $\mathbb{Z}_{2^b}$, binary SS works with $b = 1$, where $b$ is the bit length. To resist malicious clients deviating from SS specifications, we apply OT in our design (step ❹ of Protocol 4.4.3). However, this brings a considerable increase in communication costs. Furukawa *et al.* [48] used TinyOT to generalize

**Algorithm 9:** MUDGUARD.

```
Input: training dataset $\mathscr{D} = \bigcup_{i=1}^{n} \mathscr{D}_i$
Output: global models $\{w^i \mid i \in [n]\}$
1  ServerAggregation:
2  Initialize global model $w_0$
3  for each global epoch $t = 1,2,\cdots,T$ do
4      for client $i \in n$ in parallel do
5          $[[g_t^i]] \leftarrow$ ClientUpdate$(i, w_{t-1}^i)$
6      end
7      $[[G_t^j]] \leftarrow$ Algorithm 10
8      return $[[G_t^j]], \mathsf{H}_{\delta,\phi}([[G_t^j]])$ to clients
9  end
10 ClientUpdate$(i,\ w_{t-1}^i)$:
11 $\mathscr{B} \leftarrow$(split $\mathscr{D}_i$ into batches of size $b$)
12 $\mathsf{IndM} \leftarrow$MajorityVote$(\{\mathsf{IndM}_i \mid i \in [s]\})$
13 $l \leftarrow$DBSCAN$(\mathsf{IndM})$
14 reconstruct $G_{t-1}^i$ by $[[G_{t-1}^i]]$
15 if $\prod \mathsf{H}_{\delta,\phi}(\hat{g}_{t-1}^i) = \mathsf{H}_{\delta,\phi}(G_{t-1}^i)$ then
16     accept and continue
17 else
18     refuse and break
19 end
20 $w_t^i \leftarrow w_{t-1}^i - \eta \cdot \mathsf{sign}(G_{t-1}^i)$
21 $g_t^i \leftarrow$LocalTraining$(w_t^i; batch; loss)$
22 $\tilde{g}_t^i \leftarrow g_t^i / \max(1, \|g_t^i\|_2/\Delta) + \mathcal{N}(0, \Delta^2\sigma^2)$
23 $\hat{g}_t^i \leftarrow \mathsf{KS}(\tilde{g}_t^i, \mathcal{N}) \cdot \tilde{g}_t^i$
24 $\bar{g}_t^i \leftarrow \mathsf{ECD}(\mathsf{sign}(\hat{g}_t^i))$
25 send $[[\bar{g}_t^i]]$ to servers
26 broadcast $\mathsf{H}_{\delta,\phi}(\mathsf{sign}(\hat{g}_t^i))$
```

multi-party shares with communication complexity linear in the security parameter. We follow this method so that each client $i$ binary shares its updates to $S$ servers. The SS scheme guarantees that a malicious server cannot reconstruct the secret even if colluding with the rest of the servers under a malicious minority setting.

**XOR.** In step ❺ of Protocol 4.4.3, after receiving shares, each server can compute the pairwise dot product independently. Assume a server $s$ has $[[\bar{g}_t^i]]_s$, where $s \in [S]$. Since $\bar{g}_t^i = [[\bar{g}_t^j]]_1 \oplus \cdots \oplus [[\bar{g}_t^j]]_S$, we have $\bar{g}_t^i \oplus \bar{g}_t^j = [[\bar{g}_t^i]]_1 \oplus [[\bar{g}_t^j]]_1 \cdots \oplus [[\bar{g}_t^i]]_S \oplus [[\bar{g}_t^j]]_S, \forall i,j \in n$. Therefore, in this case, each server $s$ can compute $[[dot\_product]]$ by $\{[[dot\_product_{ij}]]_s = [[\bar{g}_t^i]]_s \oplus [[\bar{g}_t^j]]_s \mid \forall i,j \in [n]\}$ locally and without interactions with other servers. By multiplying a constant, one can derive shares of adjusted cosine similarity. Using binary SS can help us to save element multiplication and division operations.

**Algorithm 10:** Secure clustering.

```
Input: shares of gradients: $\{[[\vec{\mathbf{g}}_t^i]] \mid i \in [n]\}$
Output: shares of aggregation $\{[[\boldsymbol{G}_t^z]] \mid z \in c\}$
for each $i, j \in n$ do
    $[[dot\_product_{ij}]] \leftarrow [[\vec{\mathbf{g}}_t^i]] \oplus [[\vec{\mathbf{g}}_t^j]]$
    convert binary sharing $([[dot\_product_{ij}]], [[\vec{\mathbf{g}}_t^i]])$ to arithmetic sharing by B2A
    $[[\mathsf{CosM}_{ij}]] \leftarrow 1 - \frac{2}{np} \sum [[dot\_product_{ij}]]$
end
for each $i, j \in n$ do
    $[[x_{ij}]] \leftarrow ([[\mathsf{CosM}_i]] - [[\mathsf{CosM}_j]])^2$
    $[[\mathsf{EucM}_{ij}]] \leftarrow 1 + \frac{[[x_{ij}]]-1}{2} - \frac{([[x_{ij}]]-1)^2}{8} + \frac{([[x_{ij}]]-1)^3}{16}$
    if $\mathsf{EucM}_{ij} \leq \alpha$ then
        $[[\mathsf{IndM}_{ij}]] == [[1]]$
    else
        $[[\mathsf{IndM}_{ij}]] == [[0]]$
    end
end
reconstruct IndM
each server broadcasts IndM
$l \leftarrow$ DBSCAN(IndM)
for each $z \in c$ all servers in parallel do
    $[[\boldsymbol{G}_t^z]] \leftarrow \sum_{l_i = z} \mathsf{DCD}([[\vec{\mathbf{g}}_t^i]]), i \in n$
    return $[[\boldsymbol{G}_t^z]]$ to clients $\{i \mid l_i = z\}$
end
```

**Bit to Arithmetic Conversion.** The servers also need to convert the shares in $\mathbb{Z}_2$ to arithmetic shares ($\mathbb{Z}_{2^b}$) to support the subsequent linear operations and multiplications. We implement the conversion by following [49]. A common method is to use correlated randomness in these two domains (doubly-authenticated bits) and extend them. After this, the servers can derive arithmetic shares of the dot product. Note some works [50, 51] leverage straightforward transformation under the cases with only semi-honest parties.

**Multiplication.** As shown in step ❻ of Protocol 4.4.3, multiplications are necessary in DBSCAN. If we consider the semi-honest majority setting on the server side, the replicated SS and SSS can be applied here since both satisfy the multiplicative property, in which two shares multiplications can be computed locally without any interaction. For the existence of malicious servers, we consider the protocol proposed by Lindell *et al.* [52], modifying SPDZ [53] to the setting of multiplicative secret sharing modulo a prime (including replicated SS and SSS). Furukawa *et al.* [48] also proposed a similar variant for TinyOT. Both are based on the observation that the optimistic triple production using HE or OT can be replaced by producing a triple using multiplicative secret sharing instead.

**Secure Comparison with Density $\alpha$.** With arithmetic shares, the comparison (step

❼ of Protocol 4.4.3) requires extra correlated randomness, especially secret random bits in the larger domains. For the semi-honest majority servers, we follow the protocol [54] with $\mathbb{Z}_2$ to implement comparison efficiently. Under the malicious minority, we should check if the output is actually a bit. We follow [35] to multiply a secret random bit with comparison output and then reconstruct it. If the reconstructed value is a bit, it proves that the malicious servers do not deviate from the comparison protocol.

### 4.4.5. SECURITY ANALYSIS

MUDGUARD achieves security properties under *malicious majority* clients and *malicious minority* servers. Malicious parties may arbitrarily deviate from the protocol, while the rest of the parties are semi-honest, trying to infer information as much as possible (but following the protocol). We assume malicious clients and servers may collude with each other.

A secure FL system satisfies correctness, privacy, and soundness. The latter two are security requirements. Informally, the requirements are: (1) the adversary learns nothing but the differentially private output; (2) the adversary cannot provide an invalid result accepted by a benign client. We first define the ideal functionality $\mathcal{F}_{\mathsf{MUDGUARD}}$ to execute a byzantine-robust privacy-preserving FL, and then show that the proposed protocol $\Pi_{\mathsf{MUDGUARD}}$ securely realizes the functionality. Our security is based on the random oracle model where the homomorphic hash function outputs a uniformly random value for a new query and the same value for a previously answered query, hence we prove the UC security in $\mathcal{F}_{\mathsf{RO}}$-hybrid model. Besides, the security is also based on the existence of a secret sharing protocol where the clients derive shares indistinguishable with randoms which securely realizes $\mathcal{F}_{\mathsf{SS}}$ (a combination of $\mathcal{F}_{\mathsf{triples}}, \mathcal{F}_{\mathsf{share}}, \mathcal{F}_{\mathsf{reconst}}$ [48]), and a bit-to-arithmetic conversion protocol that securely realizes $\mathcal{F}_{\mathsf{B2A}}$ (noted as $\mathcal{F}_{\mathsf{PREP}}$ in [49]).

**Ideal Functionality $\mathcal{F}_{\mathsf{share}}$**

The functionality $\mathcal{F}_{\mathsf{share}}$ interacts with a dealer party $\mathsf{P}_j$, and a corrupted party $\mathsf{P}_i$.

Upon receiving $(t_i, s_i)$ from the corrupted party $\mathsf{P}_i$, and receiving $v$ from the dealer $\mathsf{P}_j$, the functionality $\mathcal{F}_{\mathsf{share}}$ computes $(t_{j+1}, s_{j+1})$ and $(t_{j+2}, s_{j+2})$ from $(t_i, s_i)$ and $v$, and sends the honest $\mathsf{P}_{i-1}$ and $\mathsf{P}_{i+1}$ their respective shares.

**Ideal Functionality $\mathcal{F}_{\mathsf{reconst}}$**

The functionality $\mathcal{F}_{\mathsf{reconst}}$ interacts with an adversary $\mathsf{Sim}$ and a corrupted party $\mathsf{P}_i$, and receives information from $\mathsf{P}_{i+1}$ and $\mathsf{P}_{i+2}$.

Upon receiving $(t_{i+1}, s_{i+1}, j)$ from $\mathsf{P}_{i+1}$ and $(t_{i+2}, s_{i+2}, j)$ from $\mathsf{P}_{i+2}$, $\mathscr{F}_{\mathsf{reconst}}$ computes $v = s_{i+2} \oplus t_{i+1}$ and sends $v$ to $\mathsf{P}_j$. In addition, the functionality $\mathscr{F}_{\mathsf{reconst}}$ sends $(t_i, s_i)$ to the adversary Sim, where $(t_i, s_i)$ is $\mathsf{P}_i$'s share as defined by the shares received from the honest parties.

**Ideal Functionality $\mathscr{F}_{\mathsf{triples}}$**

The functionality $\mathscr{F}_{\mathsf{triples}}$ interacts with a corrupted party $\mathsf{P}_i$, and receive information from $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$.

Upon receiving $N$ triples of pairs $\{(t^j_{a_i}, s^j_{a_i}), (t^j_{b_i}, s^j_{b_i}), (t^j_{c_i}, s^j_{c_i})\}_{j=1}^N$ from $\mathsf{P}_i$, the functionality first $\mathscr{F}_{\mathsf{triples}}$ chooses random $a_j, b_j \in \{0,1\}$ and computes $a_j b_j$, and then defines a vector of sharings $\boldsymbol{d} = ([a_j],[b_j],[c_j])$, for $j = 1, \ldots, N$. The sharings are computed from $[(t^j_{a_i}, s^j_{a_i}), (t^j_{b_i}, s^j_{b_i}), (t^j_{c_i}, s^j_{c_i})]$ provided by $\mathsf{P}_i$ and the chosen $a_j, b_j, c_j$. Next, $\mathscr{F}_{\mathsf{triples}}$ sends the generated shares to each corresponding party.

**Ideal Functionality $\mathscr{F}_{\mathsf{Prep}}$ ($\mathscr{F}_{\mathsf{B2A}}$)**

Independent copies of $\mathscr{F}_{\mathsf{MPC}}$ are identified via session identifiers sid. For each instance, $\mathscr{F}_{\mathsf{Prep}}$ maintains a dictionary $\mathsf{Dic}_{\mathsf{sid}}$. If a party provides input with an invalid sid, the $\mathscr{F}_{\mathsf{Prep}}$ outputs **reject** to all parties and await another message.

Upon receiving (**Init**, $\mathbb{F}$, sid) from all parties, initialize a new database of secrets $\mathsf{Dic}_{\mathsf{sid}}$ indexed by a set $\mathsf{Dic}_{\mathsf{sid}}$.Keys and store the field $\mathbb{F}$as $\mathsf{Dic}_{\mathsf{sid}}$.Field, if sid is a new session identifier. Set the flag $\mathsf{Abort}_{\mathsf{sid}} = \text{FALSE}$.

Upon receiving (**Input**, $i$, id, $x$, sid) from a party $\mathsf{P}_i$ and (**Input**, $i$, id, $\perp$, sid) from all other parties, if id $\notin \mathsf{Dic}_{\mathsf{sid}}$.Keys then insert it and set $\mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}] = x$. Then execute the procedure **Wait**.

Upon receiving (**Add**, $\mathsf{id}_x, \mathsf{id}_y$, id, sid), set $\mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}] = \mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}_x] + \mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}_y]$ if $\mathsf{id}_x, \mathsf{id}_y \in \mathsf{Dic}_{\mathsf{sid}}$.Keys.

Upon receiving (**Mult**, $\mathsf{id}_x, \mathsf{id}_y$, id, sid), set $\mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}] = \mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}_x] \cdot \mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}_y]$, if $\mathsf{id}_x, \mathsf{id}_y \in \mathsf{Dic}_{\mathsf{sid}}$.Keys. Then execute the procedure **Wait**.

Upon receiving (**RanEle**, id, sid), set $\mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}]$ to a random element in $\mathsf{Dic}_{\mathsf{sid}}$.Field, if id $\notin \mathsf{Dic}_{\mathsf{sid}}$.Keys. Then execute the procedure **Wait**.

Upon receiving (**RanBit**, id, sid), set $\mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}]$ to a random bit if id $\notin \mathsf{Dic}_{\mathsf{sid}}$.Keys. Then execute the procedure **Wait**.

Upon receiving (**Open**, i ,id, sid) from all parties, if id $\in \mathsf{Dic}_{\mathsf{sid}}$.Keys: 1) if $i = 0$, send $\mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}]$ to the adversary and executes **Wait**. If the answer is (**OK**, sid), await an error $\epsilon$ from the adversary. Send $\mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}] + \epsilon$ to all honest parties. If $\epsilon \neq 0$, set the flag $\mathbf{Abort}_{\mathsf{sid}} = \text{TRUE}$. 2) if $i \in A$, then send $\mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}]$ to the adversary. Then execute **Wait**. 3) if $i \in [n] \setminus A$, execute **Wait**. If not already halted, then await an error $\epsilon$ from the adversary. Send $\mathsf{Dic}_{\mathsf{sid}}[\mathsf{id}]+\epsilon$ to party $\mathsf{P}_i$. If $\epsilon \neq 0$, set the flag $\mathbf{Abort}_{\mathsf{sid}} = \text{TRUE}$.

Upon receiving $(\mathbf{Check}, \mathsf{sid})$ from all parties, execute the procedure **Wait**. If not already halted and $\mathbf{Abort}_{\mathsf{sid}} = \text{TRUE}$, send $(\mathbf{Abort}, \mathsf{sid})$ to the adversary and all honest parties, and ignore further messages to $\mathcal{F}_{\mathsf{MPC}}$ with the same sid. Otherwise, send $(\mathbf{OK}, \mathsf{sid})$ and continue.

Upon receiving $(\mathbf{daBits}, \mathsf{id}_1, \ldots, \mathsf{id}_l, \mathsf{sid}_1, \mathsf{sid}_2)$ from all parties where $\mathsf{id}_i \notin \mathsf{Dic}_{\mathsf{sid}}.\mathsf{Keys}$ for all $i \in [l]$, await a message **OK** or **Abort** from the adversary. If **OK** is received, sample a set of random bit $\{b_j\}_j \in [l]$, and for each $j \in [l]$ set $\mathsf{Dic}_{\mathsf{sid}_1}[\mathsf{id}_j] = b_j$ and $\mathsf{Dic}_{\mathsf{sid}_2}[\mathsf{id}_j] = b_j$, and insert the set $\{\mathsf{id}_i\}_{i\in[l]}$ into $\mathsf{Dic}_{\mathsf{sid}_1}.\mathsf{Keys}$ and $\mathsf{Dic}_{\mathsf{sid}_2}.\mathsf{Keys}$. Otherwise, send $(\mathbf{Abort}, \mathsf{sid}_1)$ and $(\mathbf{Abort}, \mathsf{sid}_2)$ to the adversary and all honest parties, and ignore all further messages to $\mathcal{F}_{\mathsf{MPC}}$) with the same $\mathsf{sid}_1$ and $\mathsf{sid}_2$.

Procedure **Wait**: Await a message $(\mathbf{OK}, \mathsf{sid})$ or $(\mathbf{Abort}, \mathsf{sid})$ from the adversary. If **OK** is received, then continue. Otherwise, send $(\mathbf{Abort}, \mathsf{sid})$ to all honest parties, and ignore all further messages to $\mathcal{F}_{\mathsf{MPC}}$ with the same sid.

**Remark 4.4.2.** *We use $\mathcal{F}_{share}$ as the secret share generating algorithm, $\mathcal{F}_{reconst}$ as the reconstructing algorithm, $\mathcal{F}_{triples}$ as the secret share multiplication algorithm, and $\mathcal{F}_{B2A}$ (see $\mathcal{F}_{PREP}$ in [49]) as the bit to arithmetic conversion algorithm.*

We formally define $\mathcal{F}_{MUDGUARD}$ as follows.

**Ideal Functionality $\mathcal{F}_{MUDGUARD}$**

The functionality $\mathcal{F}_{MUDGUARD}$ is parameterized with a DBSCAN algorithm with corresponding parameters, a local training SGD algorithm with appropriate variables, Gauss noise parameters $\Delta$ and $\sigma$, and the density parameter $\alpha$. The functionality $\mathcal{F}_{MUDGUARD}$ interacts with $n$ clients $\mathsf{P}_1, \ldots, \mathsf{P}_n$, $s$ remote servers $\mathsf{S}_1, \ldots, \mathsf{S}_s$, and an ideal adversary Sim.

Upon receiving $(\mathbf{Init}, \{w_0^i\}_{i\in[n]}, \{\boldsymbol{G}_0^i\}_{i\in[n]})$ from the adversary Sim, send $(\mathbf{Init}, w_0^i, \boldsymbol{G}_0^i)$ to each $\mathsf{P}_i$.

Upon receiving $(\mathbf{Update}, t, w_{t-1}^i, \mathcal{D}_i)$ from each honest client $\mathsf{P}_i$, calculate $w_t^i$, $\boldsymbol{g}_t^i$, $\tilde{\boldsymbol{g}}_t^i$, $\hat{\boldsymbol{g}}_t^i$, $\bar{\boldsymbol{g}}_t^i$, and store $(t, [[\bar{\boldsymbol{g}}_t^i]])$ for each server, and notify Sim with $(\mathbf{Update}, t, \mathsf{P}_i)$. If $t = T$, terminate the protocol. Later, when Sim replies with $(\mathbf{Update\text{-}data}, t)$, send $(\mathbf{Update}, t)$ to each server $\mathsf{S}_j$ for each $j \in [s]$. Upon receiving $(\mathbf{Update}, t, \{[[\bar{\boldsymbol{g}}'^i_t]]\}_{i\in\mathcal{I}})$ from Sim for all corrupted client index $i$, where $\mathcal{I} \subset [n]$, store $(t, [[\bar{\boldsymbol{g}}'^i_t]])$ for each honest server.

Upon receiving $(\mathbf{Update}, t)$ from a server $\mathsf{S}_j$, if $(t, [[\bar{\boldsymbol{g}}_t^i]])$ is stored for each $i \in [n]$ and for each server, calculate $\mathsf{IndM}$ and $\{[[\boldsymbol{G}_t^z]]\}_{z\in[c]}$ for each server. Then send $(\mathbf{Model}, t, \mathsf{IndM}, \{[[\boldsymbol{G}_t^z]]\}_{z\in[c]})$ to each server. If $\mathsf{S}_j$ is honest, upon receiving $(\mathbf{Update\text{-}model}, t)$ from the simulator Sim, send $(\mathbf{Update\text{-}model}, t, [[\boldsymbol{G}_t^z]])$ to the corresponding client. Otherwise, upon receiving (**Update-model**, $t, \{[\![\boldsymbol{G}_t'^z]\!]\}$) from the simulator Sim, send (**Update-model**, $t, [\![\boldsymbol{G}_t'^z]\!]$) to corresponding client.

Upon receiving (**Abort**) from either the adversary or any client, send $\perp$ to all parties and terminate.

**Remark 4.4.3.** *According to the ideal functionality, we could capture not only privacy but also soundness against malicious corruption of servers. However, the differential attack is based on the output of each epoch, which is published roundly and could be obtained legally. Hence, the discussion on differential privacy is not included in this security definition. Detailed proof for differential privacy will be given later. Moreover, soundness against malicious corruption of clients is also not captured by the previous definition since such security is protected by the clustering technique, which is not the concern of cryptography.*

**Definition 4.4.4** (Universally Composable security)**.** *A protocol $\Pi$ UC-realizes ideal functionality $\mathscr{F}$ if for any PPT adversary $\mathscr{A}$ there exists a PPT simulator $\mathscr{S}$ such that, for any PPT environment $\mathscr{E}$, the ensembles $\mathsf{EXEC}_{\Pi,\mathscr{A},\mathscr{E}}$ and $\mathsf{EXEC}_{\mathsf{IDEAL}_{\mathscr{F}},\mathscr{S},\mathscr{E}}$ are indistinguishable.*

**Definition 4.4.5** (UC security of `MUDGUARD`)**.** *A protocol $\Pi_{MUDGUARD}$ is UC-secure if $\Pi_{MUDGUARD}$ UC-realizes $\mathscr{F}$, against malicious-majority clients and malicious-minority servers, considering arbitrary collusion between malicious parties.*

**Theorem 4.4.6** (UC security of `MUDGUARD`)**.** *Suppose the existence of a homomorphic hash function in a random oracle model, our protocol is UC-secure in $(\mathscr{F}_{RO}, \mathscr{F}_{SS}, \mathscr{F}_{B2A})$-hybrid world.*

*Proof.* We show the validity of the theorem by proving that the protocol $\Pi_{MUDGUARD}$ securely realizes $\mathscr{F}$ in the $(\mathscr{F}_{\mathsf{RO}}, \mathscr{F}_{\mathsf{SS}}, \mathscr{F}_{\mathsf{B2A}})$-hybrid world against any corruption pattern. We construct a simulator Sim for any non-uniform PPT environment $\mathscr{E}$ such that $\mathsf{EXEC}^{\mathscr{F}_{\mathsf{RO}},\mathscr{F}_{\mathsf{SS}},\mathscr{F}_{\mathsf{B2A}}}_{\Pi_{MUDGUARD},\mathscr{A},\mathscr{E}} \approx \mathsf{EXEC}_{\mathscr{F}_{MUDGUARD},\mathsf{Sim},\mathscr{Z}}$. The Sim is constructed as follows.

It writes on $\mathscr{A}$'s input tape upon receiving an input value from $\mathscr{E}$, as if coming from $\mathscr{E}$, and writes on $\mathscr{Z}$'s output tape upon receiving an output value from $\mathscr{A}$, as if from $\mathscr{A}$.

*Case 1: If all clients and servers are not corrupted.* Since we assume private channels between client and server, Sim could just simply randomly choose all intermediate values. There is no distinguisher who could tell the difference between random values and real transcripts.

*Case 2: If corrupted clients exist.* We note the corrupted subset as $\mathscr{I} \subset [n]$. We need to simulate the adversary's view, which is the secret share and its corresponding hash value. For $t \in [T]$ and $i \in \mathscr{I}$, the simulator randomly chosen $\bar{\boldsymbol{g}}'^i_t \leftarrow \{0,1\}$, and internally executes $\mathscr{F}_{\mathsf{SS}}$ to obtain $[\![\bar{\boldsymbol{g}}'^i_t]\!]$. Then, Sim internally executes $\mathscr{F}_{\mathsf{RO}}$ to obtain $\mathsf{H}^i_t$. Because of the UC security of $\mathscr{F}_{\mathsf{SS}}$ and $\mathscr{F}_{\mathsf{RO}}$, there is no distinguisher that could tell the difference between $([\![\bar{\boldsymbol{g}}'^i_t]\!], \mathsf{H}^i_t)$ and $([\![\bar{\boldsymbol{g}}^i_t]\!], \mathsf{H}_{\delta,\phi}(\mathsf{sign}(\hat{g}^i_t)))$.

*Case 3: If corrupted servers exist.* We not the corrupted subset as $\mathscr{I} \subset [T]$. We need to simulate the adversary's view, including all secret shares in the protocol. It is worth noticing that we should not only guarantee the indistinguishability between two groups of shares but also the relationship among elements within each group. After obtaining $\mathsf{IndM}$, the $\mathsf{Sim}$ executes DBSCAN protocol on $\mathsf{IndM}$ and acquires cluster labels $l$, and executes the functionality $\mathscr{F}_{\mathsf{SS}}$ to obtain $[[\mathsf{IndM}_{ij}]]$ for each $i, j \in [n]$. Then, $\mathsf{Sim}$ randomly chosen $|l|$ secret sharing values $[[\bar{\boldsymbol{g}}'^{i}_{t}]]$ such that the summation $\Sigma_{l_i=z}\mathsf{DCD}([[\bar{\boldsymbol{g}}'^{i}_{t}]])$ equals to the given share $[[\boldsymbol{G}^{z}_{t}]]$. This procedure could be easily achieved by first randomly choosing the first $|l|-1$ values and then calculating the last value. For each $i \notin c$, the simulator $\mathsf{Sim}$ simply chooses the shares of gradients randomly since those values are irrelevant to the calculation. After acquiring all the shares of gradients $[[\bar{\boldsymbol{g}}'^{i}_{t}]]$, the simulation $\mathsf{Sim}$ pairwisely calculate $[[dot\_product'_{ij}]]$, and convert it to arithmetic sharing by executing the functionality $\mathscr{F}_{\mathsf{B2A}}$, and then calculate the adjusted cosine similarity matrix share $[[\mathsf{CosM}'_{ij}]]$. Next, as in Algorithm 2, $\mathsf{Sim}$ calculates $[[x'_{ij}]], [[\mathsf{EucM}'_{ij}]]$ for each $i, j \in [n]$. We claim that all shares that were previously generated are interdeducible, except between $[[\mathsf{EucM}'_{ij}]]$ and $[[\mathsf{IndM}'_{ij}]]$, since the latter two are the input/output pair of the element-wise comparison algorithm computed by a secure comparison algorithm in our protocol. Fortunately, the privacy of a secure comparison algorithm guarantees the indistinguishability between real and ideal input/output pairs. Hence, we claim that if there exists a distinguisher that could tell the difference between the real and ideal world, it contradicts either the UC security of $\mathscr{F}_{\mathsf{SS}}$ or the privacy of secure comparison protocol.
*Case 4: If there exists both corrupted clients and servers.* The situation, in this case, is simply the combination of Case 2 and 3 since there is no extra view needed to simulate.

In summary, for any PPT adversary $\mathscr{A}$ we could construct a $\mathsf{Sim}$, so that for any PPT environment $\mathscr{E}$, the $\mathsf{EXEC}_{\Pi,\mathscr{A},\mathscr{E}}$ and $\mathsf{EXEC}_{\mathscr{F}_{MUDGUARD},\mathscr{S},\mathscr{E}}$ are indistinguishable. □

UC framework captures attacks on input and intermediate data. On the contrary, differential privacy prevents the adversary from inferring about private information from outputs or updates, and such information might also be utilized by malicious clients. When false positives clustering exists, or malicious clients pretend to be honest, local updates have a chance to be revealed to the adversary. The following theorem shows that these updates do not leak any individual data due to differential privacy.

**Theorem 4.4.7.** *No adversary in corrupted client set $\mathscr{A}^c \subset \mathscr{C}$, where $|\mathscr{A}^c| \leq n-1$, can retrieve the individual values of honest clients.*

*Proof.* Since we apply differential privacy [18], the local updates cannot leak information regarding the inputs. According to Def. 4.2.1, the added differentially private noise guarantees that the aggregation is indistinguishable whether an individual update participates or not. Therefore, it guarantees the security of individual local updates while aggregation can be calculated. □

We prove the security in a $\mathscr{F}$-hybrid model. Our proof adopts three existing ideal functionalities: $\mathscr{F}_{\mathsf{RO}}$, $\mathscr{F}_{\mathsf{SS}}$ and $\mathscr{F}_{\mathsf{B2A}}$. The first is for the random oracle model, and the latter two are the ideal functionalities of secret sharing [48] and bit-to-arithmetic conversion [49] respectively. We have the following theorem:

**Theorem 4.4.8.** *MUDGUARD securely realizes $\mathscr{F}_{\mathsf{MUDGUARD}}$ in the $(\mathscr{F}_{\mathsf{RO}}, \mathscr{F}_{\mathsf{SS}}, \mathscr{F}_{\mathsf{B2A}})$-hybrid model, against malicious-majority clients and malicious-minority servers, considering arbitrary collusions between malicious parties.*

The remaining two properties are related to data output, which is not concerned with the cryptographic view. Specifically, DP is provided by adding noise, and soundness against malicious clients is provided by *Model Segmentation*.

Note our well-designed functionality captures as many attacks as possible. In other words, soundness against malicious clients and DP cannot be achieved under the UC model. On the one hand, recognizing malicious clients is quite a *subjective* task since they do not deviate from the protocol in cryptographic ways. There might be a benign client providing similar inputs that seem to be malicious, with a non-negligible possibility. On the other hand, the output with DP can be obtained by the adversary in our definition. Hence differential attacks should not be captured in the functionality.

## 4.4.6. Convergence Analysis

Let $M$ be the total number of clients in a semi-honest majority client cluster. Semi-honest clients and malicious clients are indexed by $\{1, \cdots, h\}$ and $\{h+1, \cdots, h+m\}$, respectively, where $M = h+m$ and $h > m$ if TNR is greater than 50%. The component $j$ of stochastic gradient and of true gradient are denoted as $\{\tilde{g}_{i,j}\}_{i=1}^{M}$ and $g_j$ respectively. An error probability is shown as follows.

**Lemma 4.4.9** (The bound of error probability with malicious clients)**.** *If the TNR (h/M) of the clustering is relatively high, then we have the error probability $\mathbb{P}\left[\mathsf{Sign}\left[\sum_{i=1}^{M}\mathsf{Sign}(\tilde{g}_{i,j})\right] \neq \mathsf{Sign}(g_j)\right] \leq \mathbb{P}_h \cdot \mathcal{O}(\sqrt{M/h})$, where $\mathbb{P}_h$ is the bound for the error probability without malicious clients.*

*Proof.* Every client is a Bernoulli trial with success probability $p_h$ for semi-honest clients and $p_m$ for malicious clients, respectively, to receive the true gradient signs. Let $Z_h$ be the number of semi-honest clients with true signs, which therefore equals the sum of $h$ independent Bernoulli trials, so we know $Z_h$ follows the binomial distribution $B(h, p_h)$. Similarly, we know the number of malicious clients with correct signs $Z_m$ follows the binomial distribution $B(m, p_m)$. Denote $q_h = 1 - p_h$ and $q_m = 1 - p_m$.

Let $Z$ be the total number of clients with true gradient signs, so $Z = Z_h + Z_m$. We use the Gaussian distribution to simplify the analysis. Notice that $B(h, p_h) \sim N(hp_h, hp_h q_h)$ and $B(m, p_m) \sim N(mp_m, mp_m q_m)$, so we get $Z \sim N\left(hp_h + mp_m, hp_h q_h + mp_m q_m\right)$. The event $\mathsf{Sign}\left[\sum_{i=1}^{M}\mathsf{Sign}(\tilde{g}_{i,j})\right] \neq \mathsf{Sign}(g_j)$ is equivalent to event $Z \leq M/2$. Then the error probability equals $\mathbb{P}(Z \leq M/2)$. By using

Cantelli's inequality, we know

$$\begin{aligned}
&\mathbb{P}[Z \le M/2] = \mathbb{P}[Z \ge 2(hp_h + mp_m) - M/2] \\
&= \mathbb{P}[Z - (hp_h + mp_m) \ge (hp_h + mp_m) - M/2] \\
&\le \frac{1}{1 + \frac{[(hp_h+mp_m)-M/2]^2}{hp_hq_h+mp_mq_m}} \le \frac{\sqrt{hp_hq_h + mp_mq_m}}{2|(hp_h + mp_m) - M/2|} \\
&= \frac{\sqrt{Mp_hq_h}}{2M(p_h - 1/2)} \cdot \frac{\sqrt{h/M + m/M \cdot p_mq_m/p_hq_h}}{(h/M \cdot p_h + m/M \cdot p_m - 1)/(p_h - 1/2)} \\
&= \mathbb{P}_h \cdot \mathcal{O}(\sqrt{M/h})
\end{aligned} \tag{4.1}$$

where the second inequality holds since $\frac{1}{x^2+1} \le \frac{1}{2x}$ for $x > 0$, and the last two equalities hold since $p_h > 1/2$ by [43] and we assume $hp_h > M/2$ with overwhelming probability for a sufficient large TNR. The assumption is reasonable because $hp_h = Mp_h > M/2$ if $TNR = 100\%$. The first factor in Eq. (4.1) is the bound for the error probability without malicious clients, so we get the error probability less than a $\mathcal{O}(\sqrt{M/h})$ factor of that in the case of without malicious clients. □

Let $\boldsymbol{L}$ and $\boldsymbol{\sigma}$ be non-negative losses and standard deviation of stochastic gradients $\tilde{g}$ respectively. $\forall x$, the objective values $f(x)$ are bounded by constants $f_*$ (i.e. $f(x) \ge f_*$). The objective value of 0-$th$ round is referred to as $f_0$. Under the above conditions, the results are the following.

**Theorem 4.4.10** (Non-convex convergence rate of MUDGUARD)**.** *If the TNR of the clustering is relatively high, then the global model generated in the semi-honest cluster converges at a rate*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1}\|g_t\|_1\right]^2 \le \frac{1}{\sqrt{N}}\left[\sqrt{\|\boldsymbol{L}\|_1}\left(f_0 - f_* + \frac{1}{2}\right) + \frac{2}{\mathcal{O}(\sqrt{h})}\|\boldsymbol{\sigma}\|_1\right]^2,$$

*where $N$ is the cumulative number of stochastic gradient calls up to round $T$ ($i.e., N = \mathcal{O}(T^2)$). Therefore, the higher the rate is, the closer the convergence speed is to the case without malicious clients.*

*Proof.* Following the results of Theorem 2 in [43], in the distributed SignSGD with a majority vote, we can get the non-convex convergence rate without malicious clients at

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1}\|g_t\|_1\right]^2 \le \frac{1}{\sqrt{N}}\left[\sqrt{\|\mathbf{L}\|_1}\left(f_0 - f_* + \frac{1}{2}\right) + \frac{2}{\sqrt{M}}\|\boldsymbol{\sigma}\|_1\right]^2 \tag{4.2}$$

from

$$|g_t|\mathbb{P}\left[\mathsf{Sign}\left[\sum_{i=1}^{M}\mathsf{Sign}(\tilde{g}_{i,j})\right] \ne \mathsf{Sign}(g_j)\right] \le \frac{\sigma_i}{\sqrt{M}}. \tag{4.3}$$

As Lemma 4.4.9 proved, in the existence of the malicious clients, we get (4.3)$\leq \frac{\sigma_i}{\sqrt{M}} \cdot \mathcal{O}(\sqrt{M/h}) = \frac{\sigma_i}{\mathcal{O}(\sqrt{h})}$. By plugging the result into (4.2), we have the convergence rate of MUDGUARD:

$$\begin{aligned}
\mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1}\|g_t\|_1\right]^2 \leq & \frac{1}{\sqrt{N}}\Bigg[\sqrt{\|\mathbf{L}\|_1}\left(f_0 - f_* + \frac{1}{2}\right) \\
& + \frac{2}{\sqrt{M}}\|\boldsymbol{\sigma}\|_1 \cdot \mathcal{O}(\sqrt{M/h})\Bigg]^2 \\
= & \frac{1}{\sqrt{N}}\Bigg[\sqrt{\|\mathbf{L}\|_1}\left(f_0 - f_* + \frac{1}{2}\right) \\
& + \frac{2}{\mathcal{O}(\sqrt{h})}\|\boldsymbol{\sigma}\|_1\Bigg]^2 .
\end{aligned} \tag{4.4}$$

□

## 4.4.7. Complexity Analysis

We use $d$ to denote the dimension of the model. $n$, $S$ and $c$ refer to the number of clients, servers and clusters, respectively.

• **Computation cost.** Each client's computation cost can be computed as binary SS with Tiny OT – $O(d)$. The server's computation cost consists of 4 parts: (1) computing pairwise XOR – $O(n^2)$; (2) bit to arithmetic conversion – $O(d)$; (3) multiplication for $L_2$ distance – $O(n^3)$; (4) comparison with $\alpha$ – $O(n^2)$; (5) calculating results of HHF based on the number of clusters $c$ – $O(nc)$. Thus, the total computation complexity of each server is $O(n^3)$.

• **Communication cost.** For a client in *MUDGUARD*, the communication cost can be divided into 2 parts: (1) sending updates to $S$ servers with binary SS and Tiny OT – $O(Sd)$; (2) broadcasting hash results of updates to the rest of parties – $O(n+S)$. Thus, we have communication complexity – $O(Sd+n)$ for each client. The servers' communication costs include (1) receiving correlated randomness and doubly-authenticated bits for converting a boolean shared matrix to arithmetic one – $O(n^2)$; (2) receiving triples for multiplications – $O(n^2)$; (3) receiving correlated randomness for element-wise comparison – $O(n^2)$; (4) sending shares and a random bit to other servers for reconstruction – $O(Sn^2)$; (5) sending aggregated shares and values of HHF to all clients – $O(nd)$. Overall, the communication cost for every server is $O(Sn^2)$. For detailed experimental results, refer to Section 4.5.2.

## 4.4.8. Adaptive attack

Recall that in Section 4.3.2, a Byzantine-robust aggregation strategy is available to attackers. Malicious clients can adapt their attacks to nullify the robustness of the system. Note that untargeted attacks (e.g., Krum and Trim attacks) solve an optimization problem to maximize the efficacy of attacks, meaning the strategies of untargeted attacks are already optimal. Therefore, we design and evaluate an adaptive backdoor attack for MUDGUARD. Specifically, the attack is formulated by adding a sub-task to the attack optimization problem. Given the fact that MUDGUARD achieves Byzantine-robustness by aggregating only benign updates as much as possible based on adjusted cosine distance, the sub-task of this attack is to try to minimize the adjusted cosine distance of malicious updates from that of benign updates. Formally, a malicious client $i$ first derives benign and malicious updates ($\boldsymbol{w}_t^i$ and $\boldsymbol{w}_t^{i'}$) with owned unpoisoned and poisoned data ($\mathscr{D}_i$ and $\mathscr{D}_i^{'}$), respectively, at the $t$-th round:

$$\boldsymbol{w}_t^i \leftarrow \boldsymbol{w}_{t-1} - \eta \nabla \mathscr{L}(\boldsymbol{w}_{t-1}, \mathscr{D}_i), \boldsymbol{w}_t^{i'} \leftarrow \boldsymbol{w}_{t-1} - \eta \nabla \mathscr{L}(\boldsymbol{w}_{t-1}, \mathscr{D}_i^{'}).$$

Then, client $i$ solves the optimization problem:

$$\underset{\boldsymbol{w}_t^{i'}}{\arg\min}\, \lambda \mathscr{L}_i(\boldsymbol{w}_{t-1}, \mathscr{D}_i^{'}) + (1-\lambda)\|\boldsymbol{w}_t^i - \boldsymbol{w}_t^{i'}\|_{COS},$$

where $\|\cdot\|_{COS}$ refers to adjusted cosine distance. $\lambda \in (0,1]$ is a hyperparameter to balance the efficacy and stealthiness of an attack. A smaller $\lambda$ makes the attack harder to be filtered, but its efficacy is less to be upheld. Section 4.5.1 gives a detailed analysis.

## 4.5. EVALUATION

We implement MUDGUARD in C++ and Python. We use MP-SPDZ library [34] to implement secure computations and Pytorch framework [55] for training. All the experiments are conducted on a cluster of machines with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz and NVIDIA 1080 Ti GPU, with 32GB RAM in a local area network. As for the cryptographic tools, all the parameters are set to a 128-bit security level.

**Datasets.** We use MNIST and FMNIST datasets for the image classification task.

• **MNIST [56].** It consists of 60,000 training samples and 10,000 testing samples, where each sample is a 28×28 gray-scale image of handwritten digital (0-9).

• **FMNIST [57].** It contains article images from Zalando and has the same size as MNIST, where each image is a 28×28 gray-scale image associated with a label from 10 classes.

• **CIFAR-10 [58].** It offers 50,000 training samples and 10,000 test samples, where each is a 32×32 color image in a label from 10 different objectives, and there are 6,000 images for each class.

**Classifiers.** We use LeNet and ResNet-18 to perform training and classification of the datasets.

• **LeNet [59].** Containing 6 layers (including 3 convolution layers, 2 pooling layers, and 1 fully connected layer), LeNet aims to train 44,426 parameters for image classification.

• **ResNet-18 [44].** It provides 18 layers with 11 million trainable parameters to train color images. We use a light vision of ResNet with approx. 2.07 million parameters and complete the experiments with the CIFAR-10 dataset.

To conduct a fair comparison against existing Byzantine-robust methods, we follow the training settings of [9, 10]. Based on the number of classes $L$, the clients are divided into $L$ groups. Non-iid degree $q$ determines the heterogeneity of data distribution. For example, if we use MNIST with 10 classes and $q = 0.5$, the samples with label "0" are allocated to the group "0" with probability 0.5 (but to other groups with probability $\frac{1-0.5}{10-1}$).

**Byzantine-attacks settings.** We consider six poisoning attacks aforementioned in Section 4.2.1. For GA, Krum, and Trim attacks, we adopt the default settings in [3]. Note that updates trained by data with low Poisoning Data Rate (PDR) could be close to the benign, bringing difficulties for MUDGUARD to distinguish them. But if we set the PDR too low, this could yield negative impacts on Attack Success Rate (ASR). For data poisoning attacks (LFA, BA, and Adaptive Attack (AA)), we set PDR to 1, aiming to achieve a relatively high ASR. To achieve a fair comparison, we follow the settings of BA [10], where a white rectangle with size 6x6 is seen as a trigger embedded on the left side of the image. Wang *et al.* [5] did not provide a dataset for FMNIST. In the experiments, we do not consider launching EA to FMNIST. To balance the main and attack tasks, we set $\lambda$ as 0.5.

**FL system settings.** Table 4.3 gives the detailed parameters. We follow the parameters setting of [1, 43], set the minibatch size to 128, and use the Adam optimizer [60] for training LeNet and ResNet-18. In the experiments, all the clients participate in the training from beginning to end. By default, we assume that there exist 100 clients splitting the training data with non-iid degree q=0.5; the proportion of malicious clients is set to $\xi$=0.6 (i.e., 60 out of 100 clients are malicious). The testing accuracy is computed over the whole testing dataset. We inject triggers into the whole testing dataset to inspect the ASR of BA. The Ardis and Southwest airplanes datasets with changed labels are used to inspect the ASR of EA in MNIST and CIFAR-10, respectively. Note that the main focus of the experiments is to examine the complexity of MUDGUARD and to check if MUDGUARD can effectively fight against Byzantine attacks. Thus, we do not further present details for client selection during each round of training, which will not affect the test of Byzantine robustness. In the clustering and robustness comparison, we define `weights-MUDGUARD` as a variant of MUDGUARD, which uses SGD to update models and takes pairwise adjusted cosine similarity of updates as inputs and $L_2$ norm as clustering metric, without applying any security tools.

### 4.5.1. Evaluation on Accuracy

We set the baseline as a "no-attack-and-defense" FL, which means it excludes the use of any cryptographic tools as well as Byzantine-robust solutions but only trains with fully honest parties. This reaches the highest accuracy and fastest convergence speed for FL training. We then set #clients participating in the baseline training equal to #semi-honest clients in the malicious existence case. We conduct each experiment for 10 independent trials and further calculate the average to achieve smooth and precise accuracy performance. We evaluate MUDGUARD's accuracy and ASR by varying the total number of clients, the proportion of malicious clients, and the degree of non-iid; and further compare the performance with the baseline.

Table 4.4 shows that, under GA, AA, BA, and EA, the testing accuracy is on par

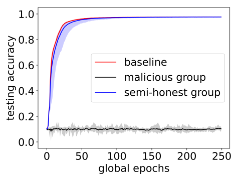
| Dataset | MNIST | FMNIST | CIFAR-10 |
|---|---|---|---|
| #clients | [10, 100, 500] | | |
| clients subsampling rate | 1 | | |
| non-iid degree | [0.1, 0.5, 0.9] | | |
| #local epochs | 1 | | |
| #global epochs | 250 | 1200 | |
| learning rate | 0.01 | 0.01 with $1e^{-5}$ weight decay | |
| proportion of malicious clients $\xi$ | [0.1, 0.6, 0.9] | | |
| LFA, BA, and AAs PDR | 1 | | |
| $\lambda$ | 0.5 | | |
| $\alpha$ | 1 | | |
| #edge-case | 300 | / | 300 |
| DPs $(\epsilon, \delta, \Delta)$ | (5, $1e^{-5}$, 5) | | |

Table 4.3: FL system settings. The parameters' range and default values are in the form of "[min, default, max]".

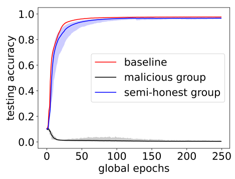
| Attacks | | baseline | GA | LFA | Krum | Trim | AA | BA | EA |
|---|---|---|---|---|---|---|---|---|---|
| $\xi$ | 0.5 | 0.975 | 0.973 | 0.967 | 0.955 | 0.965 | 0.979 / 0 | 0.972 / 0.002 | 0.966 / 0.03 |
| | 0.6 | 0.977 | 0.975 | 0.974 | 0.952 | 0.96 | 0.979 / 0.002 | 0.968 / 0.001 | 0.968 / 0.023 |
| | 0.7 | 0.975 | 0.971 | 0.971 | 0.956 | 0.953 | 0.977 / 0 | 0.963 / 0.002 | 0.953 / 0.07 |
| | 0.8 | 0.969 | 0.968 | 0.964 | 0.942 | 0.944 | 0.976 / 0.003 | 0.961 / 0.005 | 0.965 / 0.085 |
| | 0.9 | 0.969 | 0.968 | 0.968 | 0.943 | 0.937 | 0.971 / 0.005 | 0.963 / 0.002 | 0.963 / 0.093 |
| $n$ | 10 | 0.978 | 0.978 | 0.965 | 0.961 | 0.962 | 0.976 / 0 | 0.976 / 0 | 0.975 / 0 |
| | 50 | 0.975 | 0.97 | 0.958 | 0.96 | 0.949 | 0.975 / 0 | 0.975 / 0 | 0.967 / 0.02 |
| | 100 | 0.977 | 0.975 | 0.974 | 0.952 | 0.96 | 0.979 / 0.002 | 0.968 / 0.001 | 0.968 / 0.023 |
| | 200 | 0.962 | 0.962 | 0.948 | 0.951 | 0.943 | 0.963 / 0.002 | 0.961 / 0 | 0.962 / 0.042 |
| | 500 | 0.763 | 0.762 | 0.72 | 0.722 | 0.735 | 0.738 / 0.004 | 0.762 / 0.001 | 0.756 / 0.007 |
| $q$ | 0.1 | 0.976 | 0.975 | 0.978 | 0.975 | 0.975 | 0.978 / 0 | 0.975 / 0.003 | 0.976 / 0.031 |
| | 0.3 | 0.974 | 0.973 | 0.974 | 0.966 | 0.972 | 0.98 / 0 | 0.978 / 0.002 | 0.978 / 0.026 |
| | 0.5 | 0.977 | 0.975 | 0.974 | 0.952 | 0.96 | 0.979 / 0.002 | 0.968 / 0.001 | 0.968 / 0.023 |
| | 0.7 | 0.898 | 0.894 | 0.872 | 0.887 | 0.906 | 0.89 / 0.013 | 0.876 / 0.011 | 0.883 / 0.039 |
| | 0.9 | 0.709 | 0.682 | 0.705 | 0.694 | 0.689 | 0.689 / 0.017 | 0.707 / 0.025 | 0.72 / 0.06 |

Table 4.4: Comparison of accuracy with baseline and ASR by an increasing proportion of malicious clients ($\xi \geq 0.5$), #clients $n$ and non-iid degree $q$, where MNIST is used. The results under targeted attacks are in the form of testing accuracy / ASR".

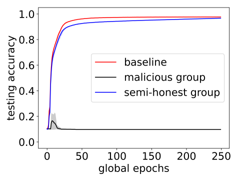
with the baseline (with only a 0.008 gap on average) in MNIST. However, compared with the baseline, the results of MUDGUARD under LFA, Krum, and Trim attacks show slight drops (on average, 0.025 in MNIST). This is so because MUDGUARD has slow convergence and large fluctuation. This is incurred by two factors. To reduce the overheads of secure computations, we apply binary SS in SignSGD. SignSGD could cause negative impacts on clustering. Only taking the signs of the gradients can ignore the effect of the magnitudes of the malicious gradients. This makes the clustering a bit prone to inaccuracy. The other factor is the LFA and Krum/Trim attacks either poison the training data and further poison updates or the local model to optimize the attacks. In the early stage of training, the malicious models do not



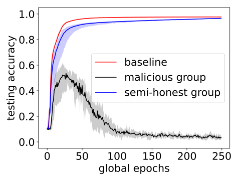
(a) Gaussian Attack



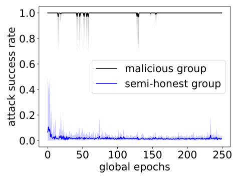
(b) LF Attack



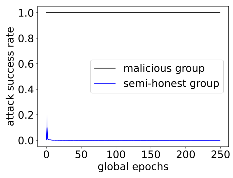
(c) Krum Attack



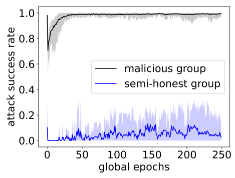
(d) Trim Attack



(e) Adaptive Attack



(f) Backdoor Attack



(g) Edge-case Attack

Figure 4.2: Comparison of testing accuracy among baseline, semi-honest, and malicious groups under untargeted attacks (a-d) and ASR between the groups under targeted attacks (e-f), where we train MNIST by the default settings in Table 4.3.

perfectly fit the poisoned training data and local models yet. Thus, the semi-honest and malicious clients could be classified into the same cluster.

Figure 4.2 presents an overview of the testing accuracy (of baseline and semi-honest and malicious groups) and ASR (of the two groups) under Byzantine attacks in the default settings of Table 4.3, where MNIST is used.[3] We see that semi-honest clients can obtain comparable accuracy to the baseline at the end of the training. In Figure 4.2a-d, the accuracy of the semi-honest group and the baseline sharply increase from 0.1 at epoch 0 to around 0.95 at epoch 25, then gradually converge to 0.97. In the GA, since the malicious group can only receive aggregation of noise, their accuracy always fluctuates around 0.1, equalling a random guess probability. As for LFA, the model accuracy gradually drops from 0.1 (at the beginning) to 0. This is because their models are trained on label-flipped datasets, while the labels of the testing set are not flipped. If the testing set is used to detect a poisoned model, the result should be flipped labels and failing to match the labels in the testing set, which results in 0. Since semi-honest and malicious clients can be classified into the same cluster at the beginning of the training, the accuracy of their models, w.r.t. malicious clients, is larger than 0.1 in some trials.

As shown in Figure 4.2b-e, the accuracy of the semi-honest group under these attacks converges slightly slower than the baseline. LFA, Krum, and Trim attacks aim to either train poisoned data or optimize local poisoned models to deteriorate the global model's testing accuracy. Due to the attacks being relatively slow and not as direct as GA, malicious updates cannot deviate 100% from benign updates at the beginning of the training (which means that malicious and semi-honest clients could

[3]The lines refer to average cases, while the shadow outlines the max and min accuracy of each epoch.

be clustered together). However, with more training rounds, the deviation becomes clearer. Thus, `MUDGUARD` separates the two groups easily.

AA, BA, and EA have no impact on the model's testing accuracy since their main purpose is to improve the ASR (nearly equal to 1 without defense). Under `MUDGUARD`, the final ASR is well suppressed. The ASR of AA and BA are close to 0 in MNIST (see Figure 4.2f-g). But the ASR of EA is much higher than that of AA and BA, reaching an average of 0.041. This is because, in EA, the edge-case training sets owned by attackers are very similar to the training sets with the target labels. If the discriminative capability of the model is not strong enough, the update directions of semi-honest and malicious gradients are also very close, making it difficult for `MUDGUARD` to distinguish them.

The experimental results in FMNIST and CIFAR-10 show the same trends as those in MNIST under the tested attacks.

| Attacks | | Baseline | GA | LFA | Krum | Trim | AA | BA |
|---|---|---|---|---|---|---|---|---|
| $\xi$ | 0.5 | 0.811 | 0.803 | 0.772 | 0.751 | 0.763 | 0.793 / 0 | 0.797 / 0 |
| | 0.6 | 0.783 | 0.772 | 0.77 | 0.761 | 0.757 | 0.784 / 0.002 | 0.801 / 0 |
| | 0.7 | 0.769 | 0.767 | 0.747 | 0.723 | 0.726 | 0.776 / 0.003 | 0.777 / 0.005 |
| | 0.8 | 0.754 | 0.752 | 0.731 | 0.743 | 0.754 | 0.771 / 0.001 | 0.758 / 0.001 |
| | 0.9 | 0.755 | 0.737 | 0.73 | 0.718 | 0.724 | 0.753 / 0.002 | 0.731 / 0.008 |
| n | 10 | 0.846 | 0.844 | 0.824 | 0.829 | 0.831 | 0.836 / 0 | 0.845 / 0 |
| | 50 | 0.834 | 0.829 | 0.829 | 0.829 | 0.827 | 0.827 / 0 | 0.836 / 0 |
| | 100 | 0.783 | 0.772 | 0.77 | 0.761 | 0.757 | 0.784 / 0.002 | 0.801 / 0 |
| | 200 | 0.774 | 0.77 | 0.763 | 0.771 | 0.766 | 0.747 / 0.002 | 0.771 / 0.002 |
| | 500 | 0.61 | 0.601 | 0.61 | 0.599 | 0.602 | 0.615 / 0.015 | 0.602 / 0.003 |
| q | 0.1 | 0.787 | 0.787 | 0.772 | 0.789 | 0.786 | 0.783 / 0 | 0.787 / 0.004 |
| | 0.3 | 0.788 | 0.777 | 0.765 | 0.773 | 0.784 | 0.783 / 0 | 0.782 / 0.002 |
| | 0.5 | 0.783 | 0.772 | 0.77 | 0.761 | 0.757 | 0.784 / 0.002 | 0.801 / 0 |
| | 0.7 | 0.65 | 0.637 | 0.657 | 0.639 | 0.65 | 0.642 / 0.008 | 0.649 / 0.006 |
| | 0.9 | 0.566 | 0.542 | 0.545 | 0.542 | 0.548 | 0.546 / 0.001 | 0.55 / 0.007 |

Table 4.5: Comparison of accuracy with baseline and ASR by an increasing proportion of malicious clients ($\xi \geq 0.5$), #clients $n$ and non-iid degree $q$, where FMNIST is used. The results under targeted attacks are in the form of testing accuracy / ASR".

In Figures 4.3 and 4.4, we present the comparison of testing accuracy among baseline, semi-honest and malicious groups under targeted attacks and ASR between the groups under untargeted attacks. In addition, we also give the experimental results by varying proportion of malicious clients ($\xi \geq 0.5$), #clients $n$ and non-iid degree $q$ in Tables 4.5 and 4.6. We see that the results are consistent with the analysis in Section 4.5.1: the untargeted attacks nearly have no impact on the accuracy of the final model (only slightly decreasing the speed of convergence). Since GA may directly upload noise, it can be easily detected from the beginning of the training to the end, resulting in the same convergence as the baseline. For LFA, Krum, Trim, and AA Attacks, `MUDGUARD` is also difficult to distinguish between semi-honest and malicious clients at the beginning. Thus, the speed of convergence

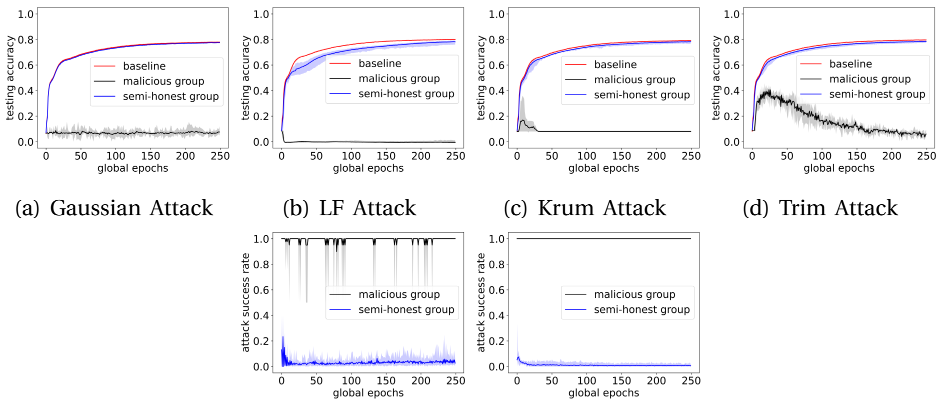
| Attacks | | Baseline | GA | LFA | Krum | Trim | AA | BA | EA |
|---|---|---|---|---|---|---|---|---|---|
| $\xi$ | 0.5 | 0.573 | 0.57 | 0.574 | 0.557 | 0.562 | 0.568 / 0.006 | 0.572 / 0.007 | 0.571 / 0.019 |
| | 0.6 | 0.562 | 0.559 | 0.559 | 0.547 | 0.534 | 0.521 / 0.011 | 0.567 / 0 | 0.568 / 0.03 |
| | 0.7 | 0.54 | 0.52 | 0.506 | 0.513 | 0.515 | 0.531 / 0.011 | 0.524 / 0 | 0.531 / 0.031 |
| | 0.8 | 0.519 | 0.508 | 0.489 | 0.494 | 0.488 | 0.482 / 0.003 | 0.52 / 0.004 | 0.501 / 0.05 |
| | 0.9 | 0.489 | 0.492 | 0.474 | 0.45 | 0.483 | 0.475 / 0.006 | 0.484 / 0.008 | 0.478 / 0.059 |
| n | 10 | 0.677 | 0.672 | 0.668 | 0.665 | 0.668 | 0.658 / 0 | 0.659 / 0.001 | 0.66 / 0.037 |
| | 50 | 0.641 | 0.637 | 0.635 | 0.653 | 0.634 | 0.639 / 0 | 0.647 / 0.001 | 0.641 / 0.068 |
| | 100 | 0.562 | 0.559 | 0.559 | 0.547 | 0.534 | 0.521 / 0.011 | 0.567 / 0 | 0.568 / 0.03 |
| | 200 | 0.46 | 0.453 | 0.474 | 0.457 | 0.45 | 0.468 / 0.003 | 0.446 / 0 | 0.458 / 0.028 |
| | 500 | 0.27 | 0.26 | 0.254 | 0.274 | 0.252 | 0.276 / 0.004 | 0.262 / 0 | 0.242 / 0 |
| q | 0.1 | 0.573 | 0.574 | 0.566 | 0.562 | 0.554 | 0.555 / 0 | 0.572 / 0.001 | 0.558 / 0.058 |
| | 0.3 | 0.567 | 0.567 | 0.556 | 0.535 | 0.553 | 0.561 / 0 | 0.569 / 0 | 0.543 / 0.064 |
| | 0.5 | 0.562 | 0.559 | 0.559 | 0.547 | 0.534 | 0.521 / 0.011 | 0.567 / 0 | 0.568 / 0.03 |
| | 0.7 | 0.426 | 0.417 | 0.394 | 0.435 | 0.424 | 0.44 / 0.013 | 0.41 / 0.004 | 0.429 / 0.015 |
| | 0.9 | 0.229 | 0.227 | 0.216 | 0.224 | 0.229 | 0.219 / 0.024 | 0.217 / 0.013 | 0.214 / 0.018 |

Table 4.6: Comparison of accuracy with baseline and ASR by an increasing proportion of malicious clients ($\xi \geq 0.5$), #clients $n$ and non-iid degree $q$, where CIFAR-10 is used. The results under targeted attacks are in the form of testing accuracy / ASR".

is slightly decreased. The main difference is that: LeNet achieves around 78% in FMNIST, while ResNet-18 provides approx. 56% accuracy in CIFAR-10.

We also provide comparisons with the state-of-the-art Byzantine-robust methods in Figure 4.5 and 4.6 on FMNIST and CIFAR-10 respectively. Similar to Figure 4.7, under the malicious majority of untargeted attacks, the testing accuracies of `FLTrust`, `MUDGUARD`, and `weights-MUDGUARD` are maintained at the same level of the baseline. Under the targeted attacks, `FLTrust`, `MUDGUARD`, and `weights-MUDGUARD` can restrain ASR to about 0%-10%. For more detailed explanations, please refer to Section 4.5.1.

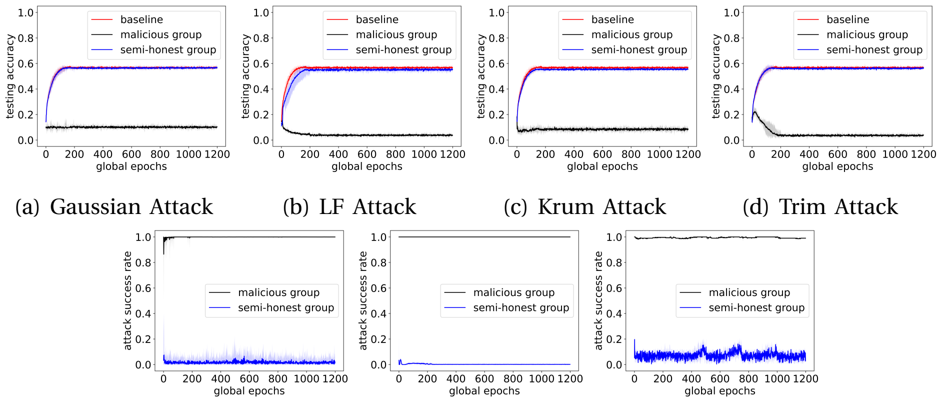
**Impact of the proportion of malicious clients.** We evaluate testing accuracy and ASR when the proportion of malicious clients $\xi \geq 0.5$. In Tables 4.4, 4.5, and 4.6, we can see that all accuracy results show a slightly downward trend with the increase of $\xi$ in three datasets. For the baseline, the accuracy on average drops 0.008, 0.057, and 0.084 in MNIST, FMNIST, and CIFAR-10, respectively. Under GA, AA, BA, and EA, this kind of decline is on par with the baseline, whether in MNIST (0.003-0.009), FMNIST (0.059-0.66), or CIFAR-10 (0.078-0.093). Under the LFA, Krum, and Trim attacks, affected by the slow convergence and fluctuation, the testing accuracy of `MUDGUARD` also declines a bit more than the baseline, which is 0.012-0.028, 0.035-0.055, and 0.089-0.107 in MNIST, FMINST, and CIFAR-10, respectively. Recall that malicious clients hold a portion of the benign dataset but do not contribute to the global model (note this equals to the case where the portion of the benign dataset is missing). From this perspective, the accuracy should be related to the number of semi-honest clients, where the max accuracy we achieve could correspond to the case when the clients are all semi-honest. Beyond the accuracy, the ASR of EA has an upward trend while the number of malicious clients is increasing, rising by



(a) Gaussian Attack (b) LF Attack (c) Krum Attack (d) Trim Attack

(e) Adaptive Attack (f) Backdoor Attack

Figure 4.3: Comparison of testing accuracy among baseline, semi-honest, and malicious groups under targeted attacks (a-e) and ASR between the groups under untargeted attack (f), where we use LeNet to train FMNIST by default settings in Table 4.3.

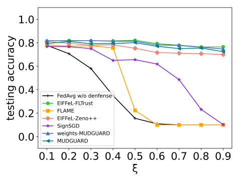


(a) Gaussian Attack (b) LF Attack (c) Krum Attack (d) Trim Attack

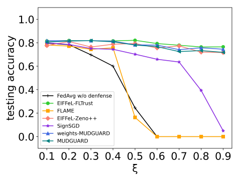
(e) Adaptive Attack (f) Backdoor Attack (g) Edge-case Attack

Figure 4.4: Comparison of testing accuracy among baseline, semi-honest, and malicious groups under targeted attacks (a-e) and ASR between the groups under untargeted attacks (f-g), where we use ResNet-18 to train CIFAR-10 by the default settings in Table 4.3.

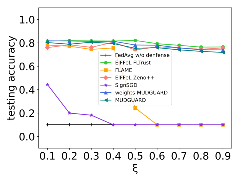
0.005 and 0.048 in MNIST and FMNIST. Since EA is not perfectly distinguished by `MUDGUARD`, the ASR naturally grows with the increase in the number of malicious clients. *In conclusion, `MUDGUARD` is effective in maintaining accuracy even when the proportion of malicious clients is ≥ 50%. While there is a slight decline in accuracy in some cases, it is on par with the baseline and does not significantly affect the overall*



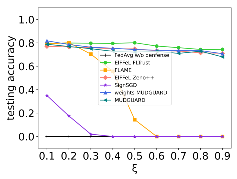
(a) Gaussian Attack



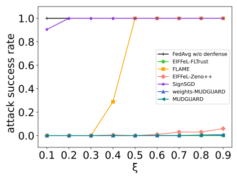
(b) LF Attack



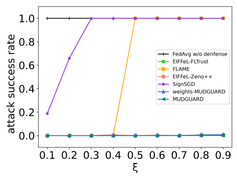
(c) Krum Attack



(d) Trim Attack

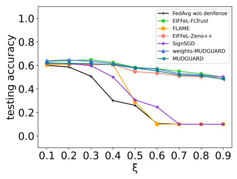


(e) Adaptive Attack

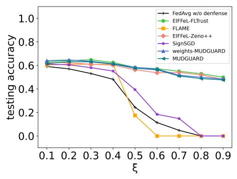


(f) Backdoor Attack

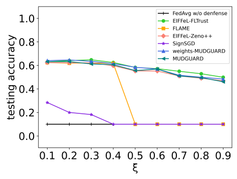
Figure 4.5: Comparison with Byzantine-robust methods in FMNIST by $\xi = 0.1 - 0.9$.



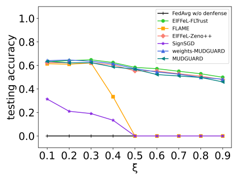
(a) Gaussian Attack



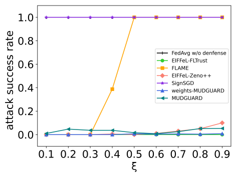
(b) LF Attack



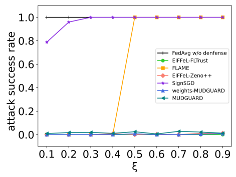
(c) Krum Attack



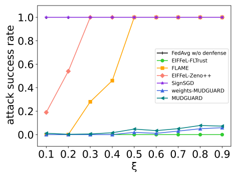
(d) Trim Attack



(e) Adaptive Attack



(f) Backdoor Attack



(g) Edge-case Attack

Figure 4.6: Comparison with Byzantine-robust methods in CIFAR-10 by $\xi = 0.1 - 0.9$.

*performance of the system.*

**Impact of the total number of clients.** Tables 4.4, 4.5 and 4.6, show the comparable testing accuracy of MUDGUARD under different attacks, as well as ASR of BA and EA when the total number of clients is set from 10 to 500. We observe that the accuracy appears to fall whilst the client number is increasing, especially when #clients = 500, it descends by about 0.2, 0.25, and 0.4 in MNIST, FMINST, and CIFAR-10, respectively. This is caused by a relatively small number of training samples. For example, in CIFAR-10, each client can only be assigned 100 samples, which does not capture one minibatch size, resulting in a “bad" performance in terms of testing accuracy. However, MUDGUARD is not affected by this factor, and it can further defend against all untargeted attacks to maintain accuracy at the same level as the baseline.

The ASR of AA and BA are controlled to nearly 0%. Although EA provides a higher ASR (than AA and BA), it drops to nearly 0 when #clients = 500, which confirms that its effectiveness relies on how well the model learns. *Overall, MUDGUARD can maintain a high level of accuracy under different attacks and across a range of client numbers and can effectively defend against untargeted attacks. Even when against EA, MUDGUARD can reduce its ASR to nearly 0%.*

**Impact of the degree of non-iid.** We further present the testing accuracy and ASR for the cases where the degree of non-iid ranges from 0.1 to 0.9 in Tables 4.4, 4.5, and 4.6. We can see that in the presence of attacks, MUDGUARD can still remain at the same level of performance as the baseline, dropping only 0.018 on average. The largest decrease is 0.067 when $q = 0.5$, which happens under the Krum attack on training LeNet with FMNIST. Note the accuracy and the degree of non-iid show a negative correlation with/without attacks, which is also in line with the conclusion of [1] that FedAvg performs not well in the case of heterogeneous data distribution. The ASR of BA appears to have a slight growth as $q$ ascends in (F)MNIST. This is because, in the high degree of non-iid, the distances among semi-honest clients also raise. For targeted attacks like AA and BA, the directions of updates are closer to those of benign updates than those of untargeted attacks. At the beginning of training, there are cases when the distances between malicious clients and semi-honest clients are similar to those between semi-honest clients, making it difficult for MUDGUARD to capture subtle differences. For the ASR of EA, as concluded in analyzing the impact of total clients, EA performs poorly when the model's accuracy is low. *As a general conclusion, MUDGUARD achieves a high Byzantine robustness. Semi-honest clients can get accurate models, while malicious clients fail to attack but also are unable to get the models.*

**Effectiveness of clustering.** To investigate the effectiveness of our clustering approach, we present the impact on True Positives Rate (TPR) and True Negatives Rate (TNR) under all attacks of $\xi = 0.6$ in Table 4.7 and compare against the method of FLAME. The FLAME takes updates as inputs and cosine similarity as a metric for clustering. Note on the server side, *Model Segmentation* does not need to identify which cluster is malicious/semi-honest. We consider false positives to occur if semi-honest clients are grouped with the malicious. On average, under GA, the TPR and TNR improve from 0.151 and 0.126 in FLAME to 1 in weights-MUDGUARD, respectively. Since MUDGUARD is based on SignSGD, only the signs of updates are taken. Ignoring the magnitude effect, there is a reduction in TPR (an average reduction of 0.046 as compared to weights-MUDGUARD). Furthermore, TNR does not drop as we set the appropriate parameters according to Theorem 4.4.1. The same changes can be captured in the case of LFA: weights-MUDGUARD has an average increase of 0.3 and 0.324 in TPR and TNR, respectively, as compared to FLAME. Compared with weights-MUDGUARD, MUDGUARD drops by 0.04 and 0.05. We see that under other attacks (LFA, Krum, Trim, AA, BA, and EA), TPR and TNR are lower than the case under GA. Because they launch attacks on either training data or optimizing poisoned models, all updates at the beginning of training have high similarities, yielding those updates being clustered together and the cases of misclustering. The true rates of CIFAR-10 are higher than those of (F)MNIST, because we can set more

| $\xi$ =0.6 | | MNIST | | FMNIST | | CIFAR-10 | |
|---|---|---|---|---|---|---|---|
| | | TPR | TNR | TPR | TNR | TPR | TNR |
| GA | FLAME | 0.821 | 0.846 | 0.848 | 0.847 | 0.879 | 0.928 |
| | weights-MUDGUARD | 1 | 1 | 1 | 1 | 1 | 1 |
| | MUDGUARD | 0.957 | 1 | 0.94 | 1 | 0.966 | 1 |
| LFA | FLAME | 0.653 | 0.612 | 0.634 | 0.655 | 0.742 | 0.711 |
| | weights-MUDGUARD | 0.974 | 0.987 | 0.975 | 0.977 | 0.98 | 0.985 |
| | MUDGUARD | 0.929 | 0.924 | 0.927 | 0.916 | 0.943 | 0.967 |
| Krum | FLAME | 0.587 | 0.622 | 0.521 | 0.63 | 0.527 | 0.578 |
| | weights-MUDGUARD | 0.974 | 0.953 | 0.973 | 0.968 | 0.971 | 0.966 |
| | MUDGUARD | 0.916 | 0.929 | 0.96 | 0.933 | 0.967 | 0.959 |
| Trim | FLAME | 0.691 | 0.679 | 0.699 | 0.664 | 0.646 | 0.615 |
| | weights-MUDGUARD | 0.976 | 0.964 | 0.975 | 0.965 | 0.973 | 0.988 |
| | MUDGUARD | 0.938 | 0.944 | 0.927 | 0.913 | 0.964 | 0.958 |
| AA | FLAME | 0.591 | 0.573 | 0.612 | 0.625 | 0.766 | 0.719 |
| | weights-MUDGUARD | 0.998 | 0.982 | 0.99 | 0.982 | 0.984 | 0.982 |
| | MUDGUARD | 0.971 | 0.943 | 0.941 | 0.935 | 0.943 | 0.96 |
| BA | FLAME | 0.777 | 0.763 | 0.794 | 0.83 | 0.856 | 0.897 |
| | weights-MUDGUARD | 0.957 | 0.969 | 0.965 | 0.97 | 0.963 | 0.979 |
| | MUDGUARD | 0.936 | 0.928 | 0.926 | 0.931 | 0.947 | 0.928 |
| EA | FLAME | 0.313 | 0.32 | _ | _ | 0.248 | 0.288 |
| | weights-MUDGUARD | 0.899 | 0.903 | _ | _ | 0.893 | 0.921 |
| | MUDGUARD | 0.856 | 0.876 | _ | _ | 0.827 | 0.83 |

Table 4.7: Effectiveness of clustering among FLAME method, weights-MUDGUARD, and MUDGUARD.

rounds to train ResNet-18. After the model converges, the true rates reach almost 100%. Thence, MUDGUARD obtains more correct clusters.

From the above analysis, we conclude that TNR and TPR are related to the number of training rounds, attack type, and the values of updates. Because MUDGUARD groups high similarity updates into one cluster and does not need to identify malicious/semi-honest clusters, the performance of clustering is less affected by the proportion of malicious clients. Similar results, like Table 4.7, can be captured even in the case when $\xi$ >0.6. Through Figure 4.2, Table 4.7, and the above discussion, we state that although TNR and TPR are affected to a certain extent by binary SS, from the view of testing accuracy and ASR, MUDGUARD achieves higher TPR and TNR than FLAME.

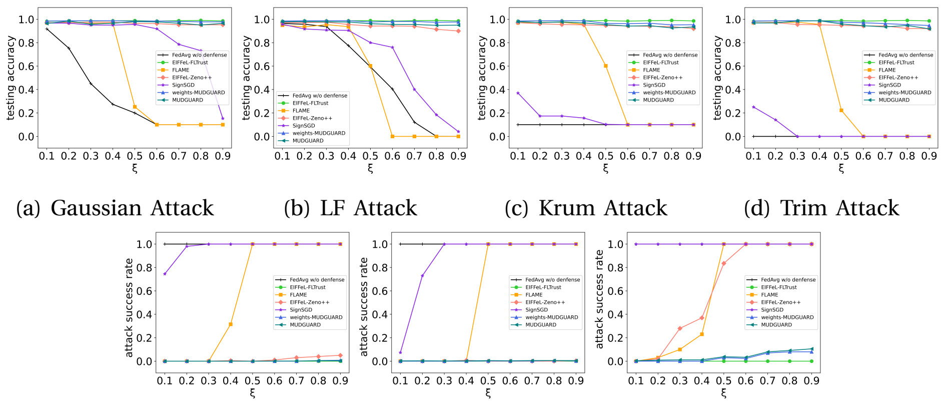
**Robustness comparison against other methods.** We present a comparison among MUDGUARD and SOTA methods (FLTrust, FLAME, Zeno++, and EIFFeL) in terms of robustness, as shown in Figure 4.7, where MNIST is used. Several Byzantine-robust

FL systems can easily and directly apply to EIFFeL. We select the two of them (please refer to EIFFeL [12]) for comparison, namely FLTrust and Zeno++. For brevity, we refer to them as EIFFeL-FLtrust and EIFFeL-Zeno++ hereafter. To demonstrate the advantages of MUDGUARD (based on SignSGD), we also compare its robustness with both SignSGD and FedAvg w/o defense. To investigate the impact of the cryptographic tools on testing accuracy and ASR, we also compare MUDGUARD with weights-MUDGUARD. One may see that MUDGUARD, countering the case of the malicious majority on the client side, does outperform most existing approaches.

In Figure 4.7a-d, the accuracy of (weights-)MUDGUARD and EIFFeL-(FLTrust/Zeno++) can be maintained at the same level as the baseline (about 0.97). Due to the impacts of misclustering, weights-MUDGUARD has a 0.02 accuracy gap with EIFFeL-FLTrust. MUDGUARD (with DP noise) commits a roughly 0.01 accuracy loss as compared to weights-MUDGUARD. The accuracy of others decreases with the increase in malicious clients, especially when $\xi \geq 0.5$, the accuracy drops abruptly to the same level of FedAvg without defense. For the ASR of AA and BA, apart from EIFFeL-(FLTrust/Zeno++), MUDGUARD and weights-MUDGUARD, all the remaining methods suddenly increase to 1 at $\xi$=0.4/0.5. Since EA has better attack ability (than AA and BA), weights-MUDGUARD and MUDGUARD suffer from a nearly 0.08 gap to EIFFeL-FLTrust. The ASR of others can raise from $\xi = 0.1$ and finally reach 1.0 at $\xi = 0.5$. SignSGD only limits the magnitude of malicious updates rather than filtering them out. Still, it can provide a certain level of defense (Figure 4.7) when there is a low malicious proportion ($\xi$=0.1-0.2) (compared to FedAvg having an average of 0.3 higher testing accuracy under untargeted attacks, and an average lower ASR of 0.4 under targeted attacks). As the number of malicious clients rises, its robustness drop to the level of FedAvg w/o defense.

FLAME indicates that a small-size cluster should be a malicious group. Thus, it is easy to confirm malicious clients via clustering. In the case of the malicious majority, it is hard to identify the malicious/semi-honest via group size. FLTrust assumes that before training, an honest server collects and trains on a small dataset. In each round, the server takes the updates trained by this small dataset as the root of trust. The “trusted" results are then compared to the updates sent by the clients. If the cosine similarity between them is too small, the updates will be filtered out. With this approach, the accuracy of the global model remains equivalent to that of the baseline. We state that MUDGUARD is on par with FLTrust, but it does not suffer from the restriction that the servers need to collect an auxiliary dataset ahead of training. We also see that when the proportion of malicious clients rises, the accuracy of MUDGUARD shows a slight decline. When clients upload their updates, MUDGUARD can only aggregate them with similar directions. If there is only a small percentage of semi-honest clients in the system, we naturally have an incomplete training set, causing a loss in accuracy. Note the same trends as those in MNIST can be seen in FMNIST (Figure 4.5) and CIFAR-10 (Figure 4.6).

**Impact of $\lambda$ on Adaptive Attack.** Figure 4.8 shows how the ASR varies in semi-honest and malicious groups when we adapt BA to MUDGUARD, where (F)MNIST, CIFAR-10 and default settings in Table 4.3 are used. In MNIST (Figure 4.8a), the ASR of the semi-honest group remains at a low level (nearly 0%) while that of the malicious



(a) Gaussian Attack (b) LF Attack (c) Krum Attack (d) Trim Attack

(e) Adaptive Attack (f) Backdoor Attack (g) Edge-case Attack

Figure 4.7: Comparison with Byzantine-robust methods by $\xi = 0.1 - 0.9$.

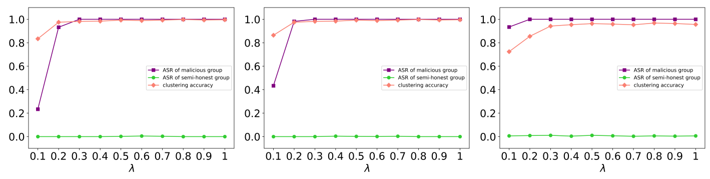
group raises from 0.23 to 1 as $\lambda$ grows from 0.1 to 0.3. This is so because when the value of $\lambda$ is low, the malicious clients using AA more focus on evading filtering (i.e., inducing a drop in clustering accuracy). Even if malicious clients are grouped with semi-honest clients, they cannot produce practical attack effectiveness. When the value of $\lambda$ gradually increases, the malicious clients will focus more on attack performance. In this way, `MUDGUARD` will easily distinguish the malicious from the semi-honest. It thus can resist AA. Note the experimental results with FMNIST and CIFAR-10 (Figure 4.8b and c) share the same trend with MNIST (Figure 4.8a).

## 4.5.2. Evaluation on Overheads

| Threat Model | Server-side | | | | Client-side | | | |
|---|---|---|---|---|---|---|---|---|
| | Semi-honest | | Malicious Minority | | Semi-honest | | Malicious Majority | |
| Traning Model | LeNet | ResNet-18 | LeNet | ResNet-18 | LeNet | ResNet-18 | LeNet | ResNet-18 |
| Runtime(Second) | 0.43±0.07/1.3±0.12 | 1.28±0.25/3.15±0.31 | 4.54±0.84/14.41±1.49 | 24.03±2.83/70.74±2.83 | 14.33±0.74 | 23.71±3.45 | 14.56±1.61 | 23.93 ±4.31 |
| Communication Costs (MB) | 16.20/53.46 | 34.82/314.45 | 873.23/2776.38 | 5151.18/15572.68 | 16.34 | 758.48 | 16.34 | 758.48 |

Table 4.8: Comparison of overheads among different threat models over LeNet & ResNet-18. The results on the server side are in the form of "optimized/unoptimized".

We conduct overheads assessment together with the evaluation of accuracy. The overheads presented in Table 4.8 capture the runtime and communication costs incurred by the implemented cryptographic tools on the server side and model training on the client side. Recall that we propose an optimization in Figure 4.1 (Section 4.4). We present the average overheads of each round of training so as to illustrate the optimized and unoptimized results in terms of different training models and honest/malicious contexts on the server side. We use LeNet and ResNet as models, and the overheads are related to their dimensionality (instead of the



(a) MNIST (b) FMNIST (c) CIFAR-10

Figure 4.8: Impact of $\lambda$ on Adaptive Attack, where backdoor attack and default settings are used.

training data).

**Runtime.** In general, we see that providing robustness in the malicious context should require more runtime than in the semi-honest. This is because extra operations for verification are taken, e.g., using HHF to verify whether a received aggregation is correct. The unoptimized ResNet-18 takes 3.15s per round in the semi-honest context while costing 70.74s (approx. an increase of 22 times) in the malicious minority. ResNet-18 has more model parameters than LeNet, leading to extra computational operations on cryptographic tools, which can be seen, in the malicious context, 70.74s v.s. 14.41s. By binary SS and polynomial transformation, Table 4.8 shows that the runtime of LeNet and ResNet-18 are reduced by 68.75% (4.54s) and 66.05% (24.03s), respectively, under malicious-minority servers.

**Communication costs.** Similar to runtime, malicious-minority servers consume a considerable amount of communication cost compared to semi-honest ones. Table 4.8 shows that after optimization, the communication costs drop to 33% in the malicious minority and 10.83% in the semi-honest with ResNet-18. In the worse case, we consume 15,572.68MB bandwidth per round under malicious minority, but we optimize the cost to 5,151.18MB. In the semi-honest context, LeNet achieves the best performance, requiring 16MB with optimization, which is 30.19% of the unoptimized cost (53.46MB).

Under the same contexts, we present the overheads of the client side for FL training in Table 4.8. Note the use of advanced FL techniques, such as those outlined in [61, 62], can be employed to enhance computing and communication efficiency in MUDGUARD. Since applying those is straightforward, we will not go into further detail on this matter.

## 4.6. Discussion

**Boosting Accuracy.** Intuitively, CIFAR-10 is more difficult to train than (F)MNIST. SignSGD [43] achieves 90% accuracy with centralized learning. The reasons for the accuracy drop in FL are (1) the number of clients: when the number increases, the amount of data allocated to a client becomes smaller, and the local model is more prone to overfitting; (2) Data heterogeneity: a larger degree of non-iid leads to less information being available for the local model to learn, which harms accuracy; (3) Malicious clients: malicious clients taking a part of the training data can be filtered out by MUDGUARD. This is equivalent to discarding that part of the training data. Additional improvements in accuracy could be obtained by using different models, e.g., ViT [63], or by performing hyperparameters grid search for FL, etc.

**Comparison of Overheads with FLAME.** Intuitively, systems that use clustering, like FLAME [10], could experience efficiency problem if left without proper optimization. In our design, MPC impacts three main phases: computation of CosM, $L_2$ distance, and element-wise comparison. The $1st$ stage is the most computationally expensive. We, therefore, focus on its optimization. After the optimization, the servers avoid using "heavy" tools, like HE and Beavers multiplication, to calculate cosine so that communication and computational overheads are naturally reduced. Therefore, MUDGUARD has much lighter complexity than FLAME.

Note that FLAME uses a 2-party semi-honest MPC scheme which is very similar to the semi-honest situation in our design (Table 4.8). Our optimized version of MUDGUARD is less computationally and communicationally complex than FLAME. In MUDGUARD, CosM is calculated by doing XOR locally on servers, and the matrix size reduces from the number of clients × the number of model updates ($n \times d$) to $n \times n$ after calculation. Then this $n \times n$ matrix is used to calculate the pairwise-$L_2$ distance (where only multiplication is involved). In FLAME, directly calculating cosine similarity requires multiplication, division, and the square root of the $n \times d$ matrix, which are relatively intensive, expensive operations in MPC. The fact that FLAME's code is not publicly available prevents a code/implementation-based comparison.

We followed state-of-the-art Byzantine-robust FL [9]'s default setting on the client number (number = 100). Table 4.8 shows overheads (the runtime and communication costs) on the server side are acceptable; in particular, only 0.4s and 16 MB are required with LeNet in the semi-honest case.

**Defending Against Other Attacks.** In the experiments, we consider SOTA (un)targeted attacks. We say that interested readers may use other attacks to test MUDGUARD, in which Byzantine-robustness could not be seriously affected. We take the Distributed Backdoor Attack (DBA) [4] as an example. DBA decomposes a global trigger into several pieces distributed to local clients. It, however, yields significant changes to some dimensions of updates to maintain the ASR of the backdoor task. Since the cosine distance between malicious and benign updates are distinguishable, MUDGUARD can still work well under DBA. Note another attack, Little Is Enough [64], have not been considered in this work because it requires attackers to have knowledge of the gradients of semi-honest clients, which violates privacy preservation.

**Advantages of Adjusted Cosine Similarity.** As shown in Figure 4.9, we present an example of the calculation of adjusted cosine similarity. It is clear to see that adjusted cosine similarity is able to capture the magnitudes and directions of updates by transferring updates to adjusted updates. Although $m_1$ does not have too many differences in directions (i.e., $m_1$ will be clustered with $h_1$ and $h_2$), its differences with $h_1$ and $h_2$ in magnitudes can be easily captured by CosM. Furthermore, due to non-iid, the ($h_1$, $h_2$) and ($h_3$, $h_4$) will be clustered into two groups if cosine distance is applied. However, by subtracting mean updates, this influence can be reduced.

We also show the experimental results in Figure 4.10, where FMNIST is used under BA and the number of clients is set to 10 (the first six are benign clients, the rest are

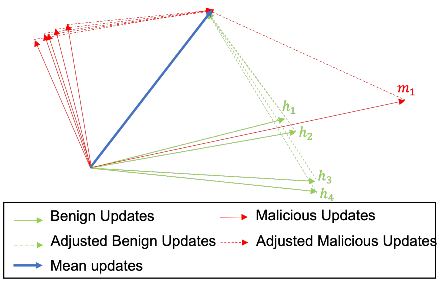
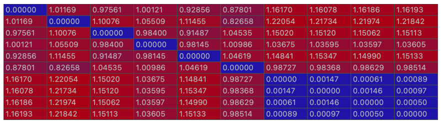


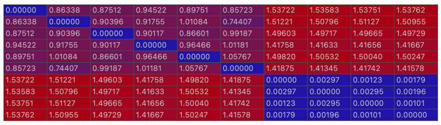
Figure 4.9: An example of calculation of adjusted cosine similarity

malicious clients). The rest of the default settings follow Table 4.3. From Figure 4.9, we can see that if only cosine distance is calculated, honest updates will be classified as noise due to the influence of non-iid. The adjusted cosine similarity weakens this effect (Figure 4.9b). Calculating the $L_2$ distance again will make the distinction between the two groups more obvious (Figure 4.9c).



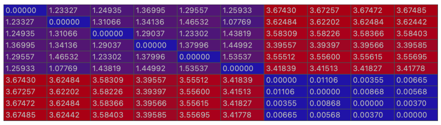
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0.00000 | 1.01169 | 0.97561 | 1.00121 | 0.92856 | 0.87801 | 1.16170 | 1.16078 | 1.16186 | 1.16193 |
| 1.01169 | 0.00000 | 1.10076 | 1.05509 | 1.11455 | 0.82658 | 1.22054 | 1.21734 | 1.21974 | 1.21842 |
| 0.97561 | 1.10076 | 0.00000 | 0.98400 | 0.91487 | 1.04535 | 1.15020 | 1.15120 | 1.15062 | 1.15113 |
| 1.00121 | 1.05509 | 0.98400 | 0.00000 | 0.98145 | 1.00986 | 1.03675 | 1.03595 | 1.03597 | 1.03605 |
| 0.92856 | 1.11455 | 0.91487 | 0.98145 | 0.00000 | 1.04619 | 1.14841 | 1.15347 | 1.14990 | 1.15133 |
| 0.87801 | 0.82658 | 1.04535 | 1.00986 | 1.04619 | 0.00000 | 0.98727 | 0.98368 | 0.98629 | 0.98514 |
| 1.16170 | 1.22054 | 1.15020 | 1.03675 | 1.14841 | 0.98727 | 0.00000 | 0.00147 | 0.00061 | 0.00089 |
| 1.16078 | 1.21734 | 1.15120 | 1.03595 | 1.15347 | 0.98368 | 0.00147 | 0.00000 | 0.00146 | 0.00097 |
| 1.16186 | 1.21974 | 1.15062 | 1.03597 | 1.14990 | 0.98629 | 0.00061 | 0.00146 | 0.00000 | 0.00050 |
| 1.16193 | 1.21842 | 1.15113 | 1.03605 | 1.15133 | 0.98514 | 0.00089 | 0.00097 | 0.00050 | 0.00000 |

(a) Pairwise cosine distance



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0.00000 | 0.86338 | 0.87512 | 0.94522 | 0.89751 | 0.85723 | 1.53722 | 1.53583 | 1.53751 | 1.53762 |
| 0.86338 | 0.00000 | 0.90396 | 0.91755 | 1.01084 | 0.74407 | 1.51221 | 1.50796 | 1.51127 | 1.50955 |
| 0.87512 | 0.90396 | 0.00000 | 0.90117 | 0.86601 | 0.99187 | 1.49603 | 1.49717 | 1.49665 | 1.49729 |
| 0.94522 | 0.91755 | 0.90117 | 0.00000 | 0.96466 | 1.01181 | 1.41758 | 1.41633 | 1.41656 | 1.41667 |
| 0.89751 | 1.01084 | 0.86601 | 0.96466 | 0.00000 | 1.05767 | 1.49820 | 1.50532 | 1.50040 | 1.50247 |
| 0.85723 | 0.74407 | 0.99187 | 1.01181 | 1.05767 | 0.00000 | 1.41875 | 1.41345 | 1.41742 | 1.41578 |
| 1.53722 | 1.51221 | 1.49603 | 1.41758 | 1.49820 | 1.41875 | 0.00000 | 0.00297 | 0.00123 | 0.00179 |
| 1.53583 | 1.50796 | 1.49717 | 1.41633 | 1.50532 | 1.41345 | 0.00297 | 0.00000 | 0.00295 | 0.00196 |
| 1.53751 | 1.51127 | 1.49665 | 1.41656 | 1.50040 | 1.41742 | 0.00123 | 0.00295 | 0.00000 | 0.00101 |
| 1.53762 | 1.50955 | 1.49729 | 1.41667 | 1.50247 | 1.41578 | 0.00179 | 0.00196 | 0.00101 | 0.00000 |

(b) Pairwise adjusted cosine distance (CosM)



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0.00000 | 1.23327 | 1.24935 | 1.36995 | 1.29557 | 1.25933 | 3.67430 | 3.67257 | 3.67472 | 3.67485 |
| 1.23327 | 0.00000 | 1.31066 | 1.34136 | 1.46532 | 1.07769 | 3.62484 | 3.62202 | 3.62484 | 3.62442 |
| 1.24935 | 1.31066 | 0.00000 | 1.29037 | 1.23302 | 1.43819 | 3.58309 | 3.58226 | 3.58366 | 3.58403 |
| 1.36995 | 1.34136 | 1.29037 | 0.00000 | 1.37996 | 1.44992 | 3.39557 | 3.39397 | 3.39566 | 3.39585 |
| 1.29557 | 1.46532 | 1.23302 | 1.37996 | 0.00000 | 1.53537 | 3.55512 | 3.55600 | 3.55615 | 3.55695 |
| 1.25933 | 1.07769 | 1.43819 | 1.44992 | 1.53537 | 0.00000 | 3.41839 | 3.41513 | 3.41827 | 3.41778 |
| 3.67430 | 3.62484 | 3.58309 | 3.39557 | 3.55512 | 3.41839 | 0.00000 | 0.01106 | 0.00355 | 0.00665 |
| 3.67257 | 3.62202 | 3.58226 | 3.39397 | 3.55600 | 3.41513 | 0.01106 | 0.00000 | 0.00868 | 0.00568 |
| 3.67472 | 3.62484 | 3.58366 | 3.39566 | 3.55615 | 3.41827 | 0.00355 | 0.00868 | 0.00000 | 0.00370 |
| 3.67485 | 3.62442 | 3.58403 | 3.39585 | 3.55695 | 3.41778 | 0.00665 | 0.00568 | 0.00370 | 0.00000 |

(c) Pairwise $L_2$ distance for CosM

Figure 4.10: Calculation results of pairwise distance.

Furthermore, we provide a comparison when MUDGUARD uses cosine similarity and adjusted cosine similarity for clustering in Figure 4.11. It is clear to see that the testing accuracy of MUDGUARD with cosine similarity abruptly goes down when the model approaches convergence. On the contrary, this does not happen in MUDGUARD with adjusted cosine similarity. The detailed explanation is given in Section 4.4.2.

**Dynamic Attacks.** Recall that MUDGUARD can perform well in terms of testing accuracy under the assumption that malicious clients consistently perform one type of attack (e.g., GA) throughout the whole training period. It can also perform well if we allow malicious clients to perform different attacks on the epochs, e.g., GA

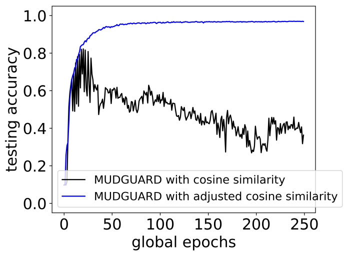


Figure 4.11: Comparison of MUDGUARD with cosine similarity and adjusted cosine similarity under GA.

to the first 10 epochs and then Krum attacks to the remaining epochs. We notice that all the attacks (we consider in this work) except GA may require several epochs of training (as a buffer) to produce attack effects. But these buffer epochs can boost MUDGUARD's clustering performance. This is so because the clustering ability is enhanced with the increase of training rounds. On the other hand, the TNR and TPR (of the clustering) could be relatively low in these epochs. Some malicious clients can be clustered into a semi-honest group, but this will not seriously affect the accuracy performance of the model.

One may think malicious clients are allowed to perform all the attacks in a single epoch. But so far, it is unknown how to group those attacks together friendly and meanwhile maximize their attack effects. In practice, the attacks may deliver an update in different directions, and further, they may even yield influence on each other. For example, GA could easily destroy the convergence of BA. We say that it is an interesting open problem to consider launching GA, LFA, Krum, Trim, and AA attacks together in an epoch to evaluate the accuracy and ARS.

**Varying Clients Subsampling Rate.** We assert that MUDGUARD can perform well under different clients' subsampling rates. This benefits from the proposed *Model Segmentation* that can resist malicious-majority clients. Imagine that in the context of an honest majority (e.g., 40 out of 100 are malicious), if the subsampling rate is set relatively low (e.g., 10%), we eventually have a malicious majority case in one training epoch with a high probability. Our experimental results have demonstrated that MUDGUARD can still achieve Byzantine-robustness under a low subsampling rate.

**Learning Rate and Local Epoch.** They are subtle parameters that decide the performance of FL training. A low learning rate can slow down the speed of convergence, and on the other hand, a high rate hinders the model's convergence, harming accuracy. As for the local epoch, in the case of iid, if carefully increasing the number of epochs, we can make the model converge fast. But under a large degree of non-iid (e.g., q=0.5), the increase of epoch leads to updates in different directions, making the FedAvg algorithm invalid [1]. In this work, we set these two parameters according to the recommendations given in [1, 43]. Exploring their impacts on training is orthogonal to the main focus of this work.

**Privacy-preserving DBSCAN.** This is one of the core parts we used to build MUDGUARD. It can apply to other real-world domains, e.g., anomaly detection and encrypted traffic analytics, where data should be clustered securely. But we note that the current MUDGUARD with optimization may not scale well in these applications. We did the optimization for the sake of efficiency by using SignSGD and binary secret sharing, which cannot support precise floating-point arithmetic.

**Test Datasets.** We notice that the EA is not applicable for FMINST since the research work [27] did not provide a backdoor dataset. We leave this as an open problem. Based on the performance of MUDGUARD on MNIST and CIFAR-10, we claim that even in FMNIST, the ASR of EA stays low, which is consistent with our main conclusion.

In the experiments, we conduct three image datasets (MNIST, FMNIST, and CIFAR-10). We claim that MUDGUARD can be further used to capture the Byzantine-robust and privacy-preserving features of the models trained on text and speech datasets, where the text dataset for the next-word prediction task can require Recurrent Neural Networks (RNNs). To train these datasets, we can also use weights or gradients to update the models (in the context of FL). If the models or datasets are poisoned (by malicious clients), malicious updates have differences in updates directions from benign ones. Then we can use MUDGUARD to protect benign clients. We could use other types of datasets during training, but this will not affect our conclusions on Byzantine robustness and privacy preservation.

**Limitations.** *Using weights as updates.* To provide cost-effective secure computations, the proposed MUDGUARD only implements the update method by SignSGD. If we use weights as updates, the secret shares sent to the servers will be in floating-point format. In this case, we will have to downgrade the design to the unoptimized MUDGUARD in Table 4.8, which could cause a considerable amount of both communication costs and runtime. An interesting future work could be to propose a more lightweight (than the current design) and secure MPC framework for MUDGUARD.

*Performance under EA.* Although MUDGUARD does achieve good performance in terms of accuracy and (to some extent) efficiency, under the EA, MUDGUARD's ASR cannot be eventually reduced to nearly 0%. In future work, we will improve the TNR and TPR of the clustering algorithm so as to recognize subtle differences between malicious and semi-honest clients.

## 4.7. CONCLUSION

We propose a novel Byzantine-robust and privacy-preserving FL system. To defend against malicious majority clients, we introduce a new approach called *Model Segmentation* and realize it using a modified DBSCAN algorithm in which we improve the accuracy of clustering by using pairwise cosine similarity. Leveraging cryptographic tools and DP, our design enables training to be performed correctly without breaching privacy. Our experimental results demonstrate that the proposed protocol effectively deals with various malicious settings for both the server and client sides and outperforms most existing solutions. Our protocols introduce reasonable overheads, which we decrease by at least 3× via appropriate optimizations.

# REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y Arcas. 'Communication-efficient learning of deep networks from decentralized data'. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.

[2] I. Dayan, H. R. Roth, A. Zhong, A. Harouni, A. Gentili, A. Z. Abidin, A. Liu, A. B. Costa, B. J. Wood, C.-S. Tsai *et al.* 'Federated learning for predicting clinical outcomes in patients with COVID-19'. In: *Nature medicine* (2021), pp. 1735–1743.

[3] M. Fang, X. Cao, J. Jia and N. Gong. 'Local Model Poisoning Attacks to {Byzantine-Robust} Federated Learning'. In: *USENIX Security*. 2020, pp. 1605–1622.

[4] C. Xie, K. Huang, P.-Y. Chen and B. Li. 'DBA: Distributed Backdoor Attacks against Federated Learning'. In: *ICLR*. 2020.

[5] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J. Sohn, K. Lee and D. S. Papailiopoulos. 'Attack of the Tails: Yes, You Really Can Backdoor Federated Learning'. In: *NIPS*. 2020.

[6] P. Blanchard, E. M. El Mhamdi, R. Guerraoui and J. Stainer. 'Machine learning with adversaries: Byzantine tolerant gradient descent'. In: *NIPS*. 2017, pp. 118–128.

[7] D. Yin, Y. Chen, R. Kannan and P. Bartlett. 'Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates'. In: *ICML*. 2018, pp. 5650–5659.

[8] E. M. E. Mhamdi, R. Guerraoui and S. Rouault. 'The hidden vulnerability of distributed learning in byzantium'. In: *ICML*. 2018, pp. 3521–3530.

[9] X. Cao, M. Fang, J. Liu and N. Z. Gong. 'FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping'. In: *NDSS*. 2021.

[10] T. D. Nguyen, P. Rieger, H. Chen, H. Yalame, H. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, S. Zeitouni, F. Koushanfar, A.-R. Sadeghi and T. Schneider. 'FLAME: Taming Backdoors in Federated Learning'. In: *USENIX Security*. 2022.

[11] C. Xie, S. Koyejo and I. Gupta. 'Zeno++: Robust fully asynchronous SGD'. In: *ICML*. 2020, pp. 10495–10503.

[12] A. Roy Chowdhury, C. Guo, S. Jha and L. van der Maaten. 'EIFFeL: Ensuring Integrity for Federated Learning'. In: *CCS*. 2022, pp. 2535–2549.

[13] K. Kumari, P. Rieger, H. Fereidooni, M. Jadliwala and A. Sadeghi. 'BayBFed: Bayesian Backdoor Defense for Federated Learning'. In: *IEEE Symposium on Security and Privacy (SP)*. 2023.

[14] L. Zhu, Z. Liu and S. Han. ‘Deep leakage from gradients’. In: *NIPS* (2019).

[15] J. Geiping, H. Bauermeister, H. Dröge and M. Moeller. ‘Inverting Gradients - How easy is it to break privacy in federated learning?’ In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 16937–16947. URL: https://proceedings.neurips.cc/paper/2020/file/c4ede56bbd98819ae6112b20ac6bf145-Paper.pdf.

[16] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang and Y. Zhou. ‘A hybrid approach to privacy-preserving federated learning’. In: *The 12th ACM Workshop on AISec*.

[17] P. Paillier. ‘Public-key cryptosystems based on composite degree residuosity classes’. In: *TAMC*. Springer. 1999.

[18] C. Dwork. ‘Differential privacy: A survey of results’. In: *TAMC*. 2008, pp. 1–19.

[19] N. Wu, F. Farokhi, D. Smith and M. A. Kaafar. ‘The value of collaboration in convex machine learning with differential privacy’. In: *IEEE S&P*. 2020, pp. 304–317.

[20] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal and K. Seth. ‘Practical secure aggregation for privacy-preserving machine learning’. In: *CCS*. 2017, pp. 1175–1191.

[21] R. Lehmkuhl, P. Mishra, A. Srinivasan and R. A. Popa. ‘Muse: Secure inference resilient to malicious clients’. In: *USENIX Security*. 2021, pp. 2201–2218.

[22] N. Koti, M. Pancholi, A. Patra and A. Suresh. ‘{SWIFT}: Super-fast and Robust {Privacy-Preserving} Machine Learning’. In: *USENIX Security*. 2021, pp. 2651–2668.

[23] I. Damgård, D. Escudero, T. Frederiksen, M. Keller, P. Scholl and N. Volgushev. ‘New primitives for actively-secure MPC over rings with applications to private machine learning’. In: *IEEE S&P*. 2019, pp. 1102–1120.

[24] D. Fiore, R. Gennaro and V. Pastro. ‘Efficiently verifiable computation on encrypted data’. In: *CCS*. 2014, pp. 844–855.

[25] B. Biggio, B. Nelson and P. Laskov. ‘Poisoning attacks against support vector machines’. In: *ICML*. 2012, pp. 1467–1474.

[26] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin and V. Shmatikov. ‘How to backdoor federated learning’. In: *AISTATS*. 2020, pp. 2938–2948.

[27] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee and D. Papailiopoulos. ‘Attack of the Tails: Yes, You Really Can Backdoor Federated Learning’. In: *NIPS*. 2020.

[28] M. Nasr, R. Shokri and A. Houmansadr. ‘Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning’. In: *IEEE S&P*. 2019, pp. 739–753.

[29] X. Luo, Y. Wu, X. Xiao and B. C. Ooi. ‘Feature inference attack on model predictions in vertical federated learning’. In: *ICDE*. 2021, pp. 181–192.

[30] A. Shamir. ‘How to share a secret’. In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[31] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.* ‘A density-based algorithm for discovering clusters in large spatial databases with noise.’ In: *kdd*. 1996, pp. 226–231.

[32] R. J. Campello, D. Moulavi and J. Sander. ‘Density-based clustering based on hierarchical density estimates’. In: *PAKDD*. 2013, pp. 160–172.

[33] J.-P. Berrut and L. N. Trefethen. ‘Barycentric lagrange interpolation’. In: *SIAM Review* 46.3 (2004), pp. 501–517.

[34] M. Keller. ‘MP-SPDZ: A versatile framework for multi-party computation’. In: *CCS*. 2020, pp. 1575–1590.

[35] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl and N. P. Smart. ‘Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits’. In: *ESORICS*. 2013, pp. 1–18.

[36] T. ElGamal. ‘A public key cryptosystem and a signature scheme based on discrete logarithms’. In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[37] C. Gentry and D. Boneh. *A fully homomorphic encryption scheme*. Vol. 20. 9. Stanford University Stanford, 2009.

[38] M. O. Rabin. ‘How To Exchange Secrets with Oblivious Transfer.’ In: *IACR Cryptol. ePrint Arch.* (2005).

[39] M. Bellare, V. T. Hoang and P. Rogaway. ‘Foundations of garbled circuits’. In: *CCS*. 2012, pp. 784–796.

[40] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar and L. Zhang. ‘Deep learning with differential privacy’. In: *The 2016 ACM CCS*.

[41] P. Mohassel and Y. Zhang. ‘SecureML: A System for Scalable Privacy-Preserving Machine Learning’. In: *IEEE S&P*. 2017, pp. 19–38.

[42] L. Bottou. ‘Large-scale machine learning with stochastic gradient descent’. In: *COMPSTAT*. 2010, pp. 177–186.

[43] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli and A. Anandkumar. ‘signSGD: Compressed optimisation for non-convex problems’. In: *ICML*. 2018, pp. 560–569.

[44] K. He, X. Zhang, S. Ren and J. Sun. ‘Deep Residual Learning for Image Recognition’. In: *CVPR*. 2016.

[45] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter and F. Koushanfar. ‘{XONN}: Xnor-based oblivious deep neural network inference’. In: *USENIX Security*. 2019, pp. 1501–1518.

[46] A. N. Bhagoji, S. Chakraborty, P. Mittal and S. Calo. ‘Analyzing federated learning through an adversarial lens’. In: *ICML*. 2019, pp. 634–643.

[47] M. Nasr, R. Shokri and A. houmansadr. *Improving Deep Learning with Differential Privacy using Gradient Encoding and Denoising*. 2020. arXiv: 2007.11524 [cs.LG].

[48] J. Furukawa, Y. Lindell, A. Nof and O. Weinstein. 'High-throughput secure three-party computation for malicious adversaries and an honest majority'. In: *EUROCRYPT*. 2017, pp. 225–255.

[49] D. Rotaru and T. Wood. *MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security*. 2019.

[50] T. Araki, A. Barak, J. Furukawa, M. Keller, Y. Lindell, K. Ohara and H. Tsuchida. 'Generalizing the SPDZ Compiler For Other Protocols'. In: *CCS*. 2018.

[51] P. Mohassel and P. Rindal. 'ABY3: A mixed protocol framework for machine learning'. In: *CCS*. 2018, pp. 35–52.

[52] Y. Lindell and A. Nof. 'A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority'. In: *CCS*. 2017, pp. 259–276.

[53] I. Damgård, V. Pastro, N. Smart and S. Zakarias. 'Multiparty Computation from Somewhat Homomorphic Encryption'. In: *CRYPTO*. 2012.

[54] A. Dalskov, D. Escudero and M. Keller. 'Fantastic Four:Honest-Majority Four-Party Secure Computation With Malicious Security'. In: *USENIX Security*. 2021, pp. 2183–2200.

[55] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.* 'Pytorch: An imperative style, high-performance deep learning library'. In: *NIPS* (2019), pp. 8026–8037.

[56] Y. LeCun and C. Cortes. 'MNIST handwritten digit database'. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[57] H. Xiao, K. Rasul and R. Vollgraf. 'Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms'. In: *arXiv preprint arXiv:1708.07747* (2017).

[58] A. Krizhevsky, G. Hinton *et al.* 'Learning multiple layers of features from tiny images'. In: (2009).

[59] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel. 'Backpropagation applied to handwritten zip code recognition'. In: *Neural computation* (1989), pp. 541–551.

[60] D. P. Kingma and J. Ba. 'Adam: A Method for Stochastic Optimization'. In: *ICLR*. 2015.

[61] J. Hamer, M. Mohri and A. T. Suresh. 'FedBoost: A Communication-Efficient Algorithm for Federated Learning'. In: *ICML*. 2020, pp. 3973–3983.

[62] A. Ghosh, J. Chung, D. Yin and K. Ramchandran. 'An Efficient Framework for Clustered Federated Learning'. In: *NIPS*. 2020, pp. 19586–19597.

[63] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.* ‘An image is worth 16x16 words: Transformers for image recognition at scale’. In: *ICLR*. 2021.

[64] G. Baruch, M. Baruch and Y. Goldberg. ‘A Little Is Enough: Circumventing Defenses For Distributed Learning’. In: *NIPS*. 2019.

# 5

# Discussion

Since the advent of Federated Learning (FL), its proprieties of privacy and security have attracted the attention of both academia and industry. In the last couple of years, FL has become one of the exciting developments where academia and industry jointly initiate real-life applications and implementations of cutting-edge technology.

Notwithstanding the interests and great efforts, FL is still a new and evolving technology, and there are numerous challenges that need to be overcome. To name a few, privacy, security, practicality, fairness, ethics, and convergence aspects with their manifold sub-challenges can be mentioned. Among the research challenges, we investigate three crucial ones for the long-term functionality of FL, which are privacy, practicality, and security aspects.

FL has been at the center of research on distributed machine learning systems as it is the first commonly accepted paradigm for legal, strategic, or economic reasons. Beyond that, many real-world applications have utilized FL, such as self-driving cars, digital health, and smart manufacturing. Therefore, our research and improvements in FL can further promote the development of FL in real-world applications.

In this chapter, we present our findings together with their limitations and potential future works. We addressed challenges listed in Section 1.2 and the research questions introduced in Section 4.3.

## 5.1. Practicality of Privacy-preserving FL

In this section, we explain the first three research questions that are related to the practicality of privacy-preserving FL and our solutions given in Chapter 2 and 3. Firstly, we investigate the design of a Homomorphic Encryption (HE)-based FL system without a TTP. Secondly, we focus on the inefficient calculations brought by HE and propose a solution that is compatible with horizontal FL.

### 5.1.1. TTP

The first problem we investigate in Chapter 2 is:

**Q1:** *What developments and considerations are involved in designing a homomorphic encryption-based federated learning system without relying on a trusted third party?*

HE's ability [1–3] to perform computations on ciphertext without the need for decryption has undoubtedly positioned it as a key player in safeguarding the privacy of clients and their datasets in FL [4] systems. The concept of encrypting model updates before transmission, allowing secure aggregation on the server without compromising individual data, is a powerful privacy-preserving approach.

However, the existing HE-based FL systems [5–8] reveal a significant drawback - reliance on a TTP for key generation and distribution. This dependence introduces complexities and potential vulnerabilities. For example, the compromise of the TTP threatens the confidentiality of the entire system. Therefore, this realization has prompted a critical examination of the current state of privacy-preserving FL and the pressing need for more resilient solutions.

In our work, by introducing a threshold encryption system, federated key generation, and model quantization techniques, we get rid of the conventional reliance on TTP for key pair generation. Our approach, allowing collaborative key generation between the server and clients and avoiding single-point failure, serves as a cornerstone for building a fully decentralized FL. The use of additive discrete logarithm-based encryption for secure model aggregation, along with verifiable secret sharing, reinforces the robustness of our system against tampering and adversarial actions.

The elimination of TTP in HE-based FL not only inspires a decentralized trend but also brings numerous positive influences to FL. For example, it can contribute to flexibility. Clients having control over their encryption keys means they have ownership and control over the security of their data. This empowerment allows them to participate in the FL scheme at any time, as they have the right to decide when their data is processed. Moreover, without a central authority managing the keys, clients can have greater confidence that their data will be used only as agreed since the data owner is only them and the privacy of their information is better protected. Clients also may perceive more concrete ownership of the FL process. This perception can contribute to a positive view of the overall FL. From the perspective of jurisdictions, privacy regulations may require explicit consent and control over personal data. Allowing participants to manage their own encryption keys aligns with these regulations, making it easier for FL initiatives to comply with legal requirements and encouraging clients to participate.

For example, imagine a Vehicle-to-Everything (V2X) scenario using HE-based FL without TTP [9]. Initially, the cars generate their encryption keys locally or collaboratively with other vehicles. This process ensures that the keys are created without the involvement of the TTP. When a car wants to join or exit the FL system, it does not need to communicate with the TTP for key generation or distribution. Instead, it can work with other vehicles that are already part of the FL network to generate the required keys. Since the TTP is not involved in every key exchange or communication event, the risk of a TTP compromise affecting the security of the entire FL system is reduced. Each vehicle has more control over its encryption keys, enhancing the overall security and privacy of the FL process.

#### Limitations and Future Work

Notwithstanding our successful resolution of the TTP problem in the context of FL using HE, it is imperative to underscore an additional inherent limitation in extant HE-based FL paradigms. Specifically, these designs exhibit suboptimal scalability, particularly when confronted with deep learning models characterized by a prodigious volume of parameters. This inherent constraint is primarily attributable to the computational demands entailed by the encryption and decryption procedures applied to the entirety of trainable model parameters. Furthermore, it is essential to acknowledge that the transmission of the fully encrypted model across communication channels engenders a considerable strain on the available network resources. Therefore, the main direction of our future work will focus on how to reduce the computational and communication complexity of FL based on encryption schemes.

### 5.1.2. Efficient Cryptographic Calculations

The second problem we analyze in Chapter 2 is the following:

**Q2:** *How to optimize cryptographic calculations to reduce computational and communication overheads?*

The exploration of preserving the privacy of FL with HE is undoubtedly an appealing research line, but the efficiency challenges highlighted in the integration of HE should be mentioned [10]. The computational and communication overheads posed by HE [11] underscores the need for innovative solutions to balance the privacy-preserving benefits of encryption with the efficiency constraints of FL systems.

In our work, we introduced an innovative approach to enhance the efficiency of HE-based FL. By employing ternary quantization of local model parameters, we introduced an approximate model aggregation method. The concept of independent ciphertext and quantized gradient aggregation, coupled with selective distribution based on predefined thresholds, strategically minimizes computational and communication overheads. We also proposed the notion of distributed partial decryption, offering another approach for optimizing HE in the context of FL. This enables subsequent research to realize that the optimization of HE-based FL extends beyond refining the encryption algorithm itself. It necessitates a comprehensive exploration of additional optimization approaches, like reduction of input size and minimizing the execution of operations on the ciphertext.

Improving the efficiency of FL can further promote scalability across heterogeneous devices in FL. Diverse workloads often involve heterogeneous devices with varying computational capabilities. Efficient HE enables FL to scale across a wide range of devices, from resource-constrained IoT hubs to more powerful servers. Besides, the flexibility in handling different model complexities is important for adapting to the specific requirements of diverse tasks. Boosting the efficiency of HE-based FL allows a flexible selection of training models between different devices, since diverse data structures may require models with varying levels of computational resources. Efficient HE allows the training of more complex models without significantly increasing the computational burden.

#### Limitations and Future Work

Although our framework demonstrates notable performance characteristics in the realm of HE-based FL, there is a critical inflection point that manifests when the encoding length descends below eight bits. At this juncture, the model's learning performance exhibits a conspicuous deterioration. Consequently, we believe that the encoding length needs to be more than 11 bits, but this necessitates the use of a brute-force recovery mechanism (decryption). While a greater encoding length undeniably augments coding precision, it concurrently engenders a substantial increase in decryption time. Our empirical analyses corroborate that global models trained with a maximum of fifteen bits can still sustain performance levels akin to non-encrypted FL paradigms, all while maintaining an acceptable recovery time. It is noteworthy that this observation is underpinned by the propensity of model gradients to asymptotically approach zero as the global model converges, thus ameliorating the decryption time incurred during brute force recovery procedures.

Nevertheless, it is incumbent upon us to recognize that the applicability of our proposed methodology is contingent upon the choice of the underlying model architecture. In contexts featuring smaller models, such as logistic regression, the inherent trade-off between model performance and computation time for plaintext recovery assumes greater prominence. Smaller models necessitate significantly longer encoding lengths, exceeding fifteen bits, thereby rendering brute force-based recovery unfeasible. In light of this, our future research endeavors are poised to concentrate on the development of a distributed additive HE framework devoid of recovery requirements, thereby rendering it amenable for seamless integration into FL systems.

### 5.1.3. Decentralized Labels in VFL

In Chapter 3, we address the following question:

**Q3:** *How to preserve the privacy of clients in the vertical federated learning system if the labels are decentralized?*

The context of privacy-preserving VFL presents a compelling journey into the intricacies of handling distributed label scenarios, a crucial but often overlooked practical challenge in the existing methodologies [12–17]. Specifically, this research line primarily assumes central control over label management, neglecting the demands of real-world scenarios where labels are distributed across clients. This necessitates the development of an innovative approach, and our endeavor focuses on secure and efficient training of XGBoost models [18] within a VFL framework.

The foundation of our solution lies in an elegant combination of secure aggregation techniques rooted in cryptographic protocols, prominently featuring Diffie-Hellman key exchange [19] and Key Derivation Function (KDF) [20]. Drawing inspiration from the Global Data Privacy (GDP) [21] framework, our approach aimes not only to address the label distribution challenge but also to preserve the privacy and integrity of both labels and features. Our work represents an advancement in the realm of privacy-preserving VFL, refining data distribution types to decentralized training labels.

Theoretical considerations are a pivotal aspect of our research, as we conducted a comprehensive security analysis within the semi-honest setting. This analysis aimed to illuminate the robustness and privacy-preserving capabilities of our solution. We also delved into the practical issue of client collusion, specifically addressing scenarios where a subset of clients conspires to subvert the integrity of the system.

Through this research, a notable impact is the stimulation of a critical rethinking of data distribution in VFL. Decentralized label settings make FL applicable to a wider range of scenarios. In real-world applications, data is often distributed across different entities, and each entity may have control over its own labels. By accommodating decentralized labels, FL becomes more versatile and suitable for diverse use cases, including healthcare, finance, and IoT. The successful integration of cryptographic protocols and the careful balance between security and efficiency in our approach underscore the viability of privacy-preserving VFL methodologies. One may gain a profound understanding of the complexities associated with privacy-preserving VFL and the critical importance of addressing real-world label distribution challenges.

Furthermore, it can contribute to improved robustness of model training. In traditional VFL with a single client holding all the training labels, the system is susceptible to disruptions if that client experiences issues or becomes unavailable. Decentralizing labels means that the failure or unavailability of one client has a smaller impact since the labels are distributed across multiple entities. This reduces the likelihood of a single point of failure and enhances the overall robustness of the FL system. Overall, this work leads a new trend for future research based on VFL.

### Limitations and Future Work

Differential privacy, while crucial for preserving the privacy of participants in FL, introduces a critical challenge that impacts the quality and utility of the federated model. This challenge stems from the necessity of adding noise either to the data or the model parameters. It is introduced to obfuscate individual contributions and protect participant privacy. However, a consequential side effect of this noise is its potential to distort the underlying, genuine information encoded in the data or model parameters. Consequently, the accuracy and overall utility of the federated model are significantly diminished. This delicate balance between safeguarding privacy and maintaining utility necessitates a nuanced examination of the trade-off inherent in the application of differential privacy within the context of FL.

For instance, empirical research has uncovered the tangible impact of differential privacy on the accuracy of FL models [22]. Notably, the introduction of differential privacy mechanisms resulted in a substantial reduction in accuracy when compared to models trained on plain, non-private data. In the realm of image classification using convolutional neural networks, these mechanisms were found to decrease accuracy by a noteworthy margin. Similarly, in the domain of natural language processing, specifically with the employment of recurrent neural networks, the application of differential privacy led to a significant accuracy reduction. These empirical findings provide compelling evidence that differential privacy indeed carries a substantial and potentially detrimental impact on the accuracy of FL.

The need for addressing this intricate balance between privacy preservation and utility enhancement in FL underscores the critical importance of developing advanced techniques that can mitigate the adverse effects of differential privacy while still ensuring robust privacy protection. Consequently, it is imperative to explore innovative methods and strategies that strike a more harmonious equilibrium between privacy and utility in the context of FL. Such endeavors will undoubtedly contribute to the advancement of privacy-preserving machine learning paradigms and their real-world applicability.

One way to improve the accuracy of FL is denoising [23]. One example of denoising is to use of a Bayesian approach with compensation to reconstruct the neural network weights from their noisy manifestations. This approach can outperform the maximum likelihood estimation in terms of inference accuracy and robustness. It can also handle different types of noise, such as additive white Gaussian noise, quantization noise, or mixed. Therefore, in the future, exploring the effectiveness of the denoising technique is vital for privacy-preserving VFL.

## 5.2. IMPROVING BYZANTINE ROBUSTNESS OF FL

In Chapter 4, we address the following question:

**Q4:** *How to capture the Byzantine robustness of honest clients in federated learning if the majority of clients are malicious?*

Reflecting on the Byzantine-robust FL [24–30], our research delved into the pervasive challenge of mitigating the impact of malicious clients. Existing methods often face a critical limitation when dealing with a malicious majority surpassing a predefined threshold. This vulnerability stems from the susceptibility of conventional consensus mechanisms to manipulation by Byzantine clients. By introducing the innovative aggregation strategy "Model Segmentation", we depart from traditional complex detection algorithms by emphasizing the separation of clients with different behaviors. This not only enhances the FL system's resilience against attacks but also simplifies the overall algorithmic complexity.

In addressing the insufficient separation effect of non-iid data distribution scenarios, we proposed an innovative methodology for updates clustering. By computing pairwise adjusted cosine similarity, we introduced a preprocessing step to enhance the accuracy of clustering. This step, seamlessly integrated into the DBSCAN algorithm [31], proves effective in capturing variations in both directionality and magnitude of updates between clients.

Another pivot in this work is the incorporation of cryptographic tools to fortify FL against malicious intent. The use of cryptographic tools, including the Homomorphic Hash Function (HHF) [32], safeguards updates on the server side and ensures correct aggregations received by all clients. Optimization strategies such as binary secret sharing and polynomial transformation further enhanced the efficiency of secure computations on the server side.

Our approach focuses on preemptive measures and leverages clustering without the need for auxiliary datasets. By abandoning the reliance on complex detection algorithms and proposing a strategic approach, our work paves the way for exploring new ideas and methodologies for defending against the malicious majority of clients.

Apart from these, our work can provide extra benefits like adaptability to adaptive attacks. The field of adversarial attacks and malicious strategies is dynamic, with new threats emerging over time. A system that can defend against a malicious majority is more adaptable to evolving threats, ensuring its long-term viability in the face of changing attack vectors and strategies.

For example, consider a financial scenario [33] and clients want to join without concerns about the behavior of other competing entities. Specifically, a financial institution decides to participate in the FL system to enhance its predictive models for fraud detection or risk assessment. Since there is intense competition in the financial sector, this financial institution is aware that some competitors may have malicious intentions. These competitors might try to manipulate or disrupt (i.e., They might try to inject backdoors, manipulate model updates, or train with poisoning data) the FL process to gain a competitive advantage. The beauty of our scheme is its ability to automatically adapt to the percentage of malicious clients for defense. Therefore, in this case, the new joining financial institution can confidently train the mode with the FL system without being overly concerned about the potential malicious actions of its competitors. The server overseeing the FL process also does not need to adopt complex client detection methods. This simplifies the server's role and reduces the computational overhead associated with monitoring and detecting malicious behaviors.

### Limitations and Future Work

In our work, which has been devised to optimize secure computations within FL, a deliberate strategy has been employed to ensure cost-effectiveness. This strategy centers on the exclusive implementation of the SignSGD [34] update method. This judicious choice significantly ameliorates the computational and communicative exigencies associated with secure aggregation. However, a pivotal consideration arises in the context of utilizing weights as updates within the paradigm of our work.

Upon contemplation of employing weights as updates, a fundamental shift occurs in the format of secret shares transmitted to the servers. Specifically, these secret shares will manifest in floating-point notation, deviating from the original architectural underpinnings of our work. Such a deviation carries notable ramifications, potentially necessitating a regressive transition to an unoptimized instantiation of our work. This transition entails a substantial outlay of communication costs and runtime, thus undermining the cardinal tenets of cost-effectiveness that are intrinsic to our framework.

As an avenue for prospective scholarly inquiry, it is pertinent to explore the prospects of engineering a more lightweight alternative, in contrast to our current work, while upholding the rigorous security postulates. This investigative trajectory can culminate in the development of a secure Multi-Party Computation (MPC) [35, 36] framework characterized by an equilibrium between computational efficiency and security within our work's context.

While our work exhibits commendable performance in terms of accuracy and, to a certain extent, efficiency, it is incumbent upon the scholarly discourse to acknowledge its comportment in the crucible of edge-case attack [28] scenarios. Regrettably, under the edge-case attack, the Attack Success Rate (ASR) of our work does not consistently converge toward a near-zero threshold. This empirical observation underscores a focal area for enhancement within our framework.

In forthcoming scholarly pursuits, our focus shall be concentrated on the refinement of both the True Negative Rate (TNR) and True Positive Rate (TPR) pertaining to the clustering algorithm integrated into our current architecture. By fortifying the discriminatory acumen of the clustering algorithm, thereby enabling it to elucidate nuanced distinctions between malevolent and semi-honest clients, we aspire to fortify our work's resilience in terms of edge-case attack incursions. This trajectory of inquiry aligns harmoniously with our overarching commitment to iteratively enhance the robustness and security of our work in response to the mutable adversarial landscape endemic to the FL milieu.

## 5.3. Conclusion

In conclusion, Federated Learning (FL) has emerged as a groundbreaking approach to machine learning, enabling collaborative model training without the need to share sensitive data, initially introduced by Google in 2016. FL has garnered substantial attention across academia and industry for its potential to revolutionize fields such as healthcare, finance, the Internet of Things, and edge computing. However, this innovative paradigm presents multifaceted challenges that demand innovative solutions. Firstly, while FL aims to safeguard client data privacy, potential leaks through model parameters or gradients remain a concern. Secondly, it is questionable in terms of practicality to bring FL in the real-world scenario. For example, the substantial communication overhead, exacerbated when employing cryptographic tools, and the computational burden imposed on FL must be mitigated. Thirdly, the presence of unpredictable or malicious clients can threaten the accuracy and security of the global model. In essence, these three challenges are essential prerequisites for achieving the functionality and practicality of a secure FL ecosystem. This thesis has focused on four pivotal challenges: privacy preservation, practicality, and Byzantine robustness.

Our work can be divided into two parts: the practicality of privacy-preserving FL and improving the Byzantine robustness of FL. Firstly, we proposed an innovative threshold encryption system that integrates federated key generation and model quantization techniques, eliminating the need for a Trusted Third Party (TTP) and enhancing resilience against clients who tamper information. Secondly, we introduce a novel approach involving ternary quantization and approximate model aggregation, significantly reducing computational and communication costs associated with threshold decryption. Thirdly, we present a secure and efficient training framework for XGBoost models in VFL, rooted in cryptographic protocols and evaluated for security and practicality. Lastly, we introduce "Model Segmentation", a proactive aggregation strategy designed to counter poisoning attacks in Byzantine-robust FL, which streamlines FL operations and enhances security. Each contribution is underpinned by rigorous empirical evaluations and, where applicable, formal security proofs within the Universal Composability (UC) framework.

# REFERENCES

[1] T. ElGamal. 'A public key cryptosystem and a signature scheme based on discrete logarithms'. In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[2] P. Paillier. 'Public-key cryptosystems based on composite degree residuosity classes'. In: *TAMC*. Springer. 1999.

[3] C. Gentry and D. Boneh. *A fully homomorphic encryption scheme*. Vol. 20. 9. Stanford University Stanford, 2009.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y Arcas. 'Communication-efficient learning of deep networks from decentralized data'. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.

[5] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang and Y. Zhou. 'A hybrid approach to privacy-preserving federated learning'. In: *The 12th ACM Workshop on AISec*.

[6] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar and H. Ludwig. 'HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning'. In: *The 12th ACM AISec*. 2019.

[7] L. Zhu, Z. Liu and S. Han. 'Deep leakage from gradients'. In: *NIPS* (2019).

[8] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos and Q. Yang. *SecureBoost: A Lossless Federated Learning Framework*. 2021. arXiv: 1901.08755 [cs.LG].

[9] Y. Li, X. Tao, X. Zhang, J. Liu and J. Xu. 'Privacy-preserved federated learning for autonomous driving'. In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (2021), pp. 8423–8434.

[10] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan and Y. Liu. 'BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning'. In: *2020 USENIX ATC*. ISBN: 978-1-939133-14-4. URL: https://www.usenix.org/conference/atc20/presentation/zhang-chengliang.

[11] J. Wu, W. Zhang and F. Luo. 'ESAFL: Efficient Secure Additively Homomorphic Encryption for Cross-Silo Federated Learning'. In: *arXiv preprint arXiv:2305.08599* (2023).

[12] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith and B. Thorne. 'Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption'. In: *arXiv preprint arXiv:1711.10677* (2017).

[13] Y. Liu, X. Zhang and L. Wang. 'Asymmetrically vertical federated learning'. In: *arXiv preprint arXiv:2004.07427* (2020).

[14] S. Yang, B. Ren, X. Zhou and L. Liu. 'Parallel distributed logistic regression for vertical federated learning without third-party coordinator'. In: *arXiv preprint arXiv:1911.09824* (2019).

[15] Y. Wu, S. Cai, X. Xiao, G. Chen and B. C. Ooi. 'Privacy Preserving Vertical Federated Learning for Tree-Based Models'. In: *Proc. VLDB Endow.* (2020), pp. 2090–2103.

[16] Y. Liu, Y. Kang, L. Li, X. Zhang, Y. Cheng, T. Chen, M. Hong and Q. Yang. 'A communication efficient vertical federated learning framework'. In: *Unknown Journal* (2019).

[17] X. Luo, Y. Wu, X. Xiao and B. C. Ooi. 'Feature inference attack on model predictions in vertical federated learning'. In: *ICDE*. 2021, pp. 181–192.

[18] T. Chen and C. Guestrin. 'XGBoost: A Scalable Tree Boosting System'. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785.

[19] W. Diffie and M. Hellman. 'New directions in cryptography'. In: *IEEE transactions on Information Theory* 22.6 (1976), pp. 644–654.

[20] H. Krawczyk and P. Eronen. *HMAC-based extract-and-expand key derivation function (HKDF)*. Tech. rep. RFC 5869, May, 2010.

[21] C. Dwork. 'Differential privacy: A survey of results'. In: *TAMC*. 2008, pp. 1–19.

[22] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek and H. V. Poor. 'Federated learning with differential privacy: Algorithms and performance analysis'. In: *IEEE Transactions on Information Forensics and Security* (2020).

[23] M. Nasr, R. Shokri and A. houmansadr. *Improving Deep Learning with Differential Privacy using Gradient Encoding and Denoising*. 2020. arXiv: 2007.11524 [cs.LG].

[24] M. Fang, X. Cao, J. Jia and N. Gong. 'Local Model Poisoning Attacks to {Byzantine-Robust} Federated Learning'. In: *USENIX Security*. 2020, pp. 1605–1622.

[25] T. D. Nguyen, P. Rieger, M. Miettinen and A.-R. Sadeghi. 'Poisoning attacks on federated learning-based IoT intrusion detection system'. In: *DISS*. 2020, pp. 1–7.

[26] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J. Sohn, K. Lee and D. S. Papailiopoulos. 'Attack of the Tails: Yes, You Really Can Backdoor Federated Learning'. In: *NIPS*. 2020.

[27] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin and V. Shmatikov. ‘How to backdoor federated learning’. In: *AISTATS*. 2020, pp. 2938–2948.

[28] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J.-y. Sohn, K. Lee and D. Papailiopoulos. ‘Attack of the Tails: Yes, You Really Can Backdoor Federated Learning’. In: *NIPS*. 2020.

[29] C. Xie, K. Huang, P.-Y. Chen and B. Li. ‘DBA: Distributed Backdoor Attacks against Federated Learning’. In: *ICLR*. 2020.

[30] Z. Sun, P. Kairouz, A. T. Suresh and H. B. McMahan. ‘Can you really backdoor federated learning?’ In: *arXiv preprint arXiv:1911.07963* (2019).

[31] R. J. Campello, D. Moulavi and J. Sander. ‘Density-based clustering based on hierarchical density estimates’. In: *PAKDD*. 2013, pp. 160–172.

[32] J.-S. Coron, Y. Dodis, C. Malinaud and P. Puniya. ‘Merkle-Damgård revisited: How to construct a hash function’. In: *Annual International Cryptology Conference*. Springer. 2005, pp. 430–448.

[33] S. Li, Y. Cheng, W. Wang, Y. Liu and T. Chen. ‘Learning to detect malicious clients for robust federated learning’. In: *arXiv preprint arXiv:2002.00211* (2020).

[34] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli and A. Anandkumar. ‘signSGD: Compressed optimisation for non-convex problems’. In: *ICML*. 2018, pp. 560–569.

[35] I. Damgård, V. Pastro, N. Smart and S. Zakarias. ‘Multiparty computation from somewhat homomorphic encryption’. In: *Annual Cryptology Conference*. Springer. 2012, pp. 643–662.

[36] M. Keller. ‘MP-SPDZ: A versatile framework for multi-party computation’. In: *CCS*. 2020, pp. 1575–1590.

# ACKNOWLEDGEMENTS

This is an optional chapter containing acknowledgements.

# CURRICULUM VITÆ

Rui Wang was born in Kaili, China on December 14, 1994. He obtained his bachelor's degree in Applied Physics from the Beijing University of Posts and Telecommunications, Beijing, China in 2017. After that, he received his master's degree in Cyber Security at the University of Southampton, the United Kingdom in 2018.

In 2019, he started as a Ph.D student at the University of Surrey, the United Kindom under the supervision of Dr. Kaitai Liang and Dr. Yunpeng Li. From September 2020, he worked at TU Delft under the supervision of Prof.dr.ir.R.L.Lagendijk and Dr. Kaitai Liang. He supervised multiple master students and bachelor student groups during their final thesis.

During his Ph.D., he worked on improving the privacy and security of Federated Learning. For this, he visited IMDEA Network Institute in Madrid, Spain, and worked under the supervision of Prof. Nikolaos Laoutaris.

# List of Publications

## Journal

4. **Wang, R.**, Ersoy, O., Zhu, H., Jin, Y. and Liang, K., 2022. Feverless: Fast and secure vertical federated learning based on xgboost for decentralized labels. IEEE Transactions on Big Data.

3. Liu, A., Li, B., Li, T., Zhou, P. and **Wang, R.**, 2022. AN-GCN: An Anonymous Graph Convolutional Network Against Edge-Perturbing Attacks. IEEE transactions on neural networks and learning systems.

2. Zhu, H., **Wang, R.**, Jin, Y. and Liang, K., 2021. Pivodl: Privacy-preserving vertical federated learning over distributed labels. IEEE Transactions on Artificial Intelligence.

1. Zhu, H.*, **Wang, R.***, Jin, Y., Liang, K. and Ning, J., 2021. Distributed additive encryption and quantization for privacy preserving federated deep learning. Neurocomputing, 463, pp.309-327.

## Conference and Workshop

5. Ghavamipour, A.R., Turkmen, F., **Wang, R.** and Liang, K., 2023, May. Federated Synthetic Data Generation with Stronger Security Guarantees. In Proceedings of the 28th ACM Symposium on Access Control Models and Technologies (pp. 31-42).

4. Tian, Y., **Wang, R.**, Qiao, Y., Panaousis, E. and Liang, K., 2023, April. FLVoogd: Robust And Privacy Preserving Federated Learning. In Asian Conference on Machine Learning (pp. 1022-1037). PMLR.

3. Xu, J., **Wang, R.**, Koffas, S., Liang, K. and Picek, S., 2022, December. More is better (mostly): On the backdoor attacks in federated graph neural networks. In Proceedings of the 38th Annual Computer Security Applications Conference (pp. 684-698).

2. Sun, L., Du, R., He, D., Zhu, S., **Wang, R.** and Chan, S., 2021, December. Feature Engineering Framework based on Secure Multi-Party Computation in Federated Learning. In 2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys) (pp. 487-494). IEEE.

1. Tjiam, K., **Wang, R.**, Chen, H. and Liang, K., 2021, November. Your smart contracts are not secure: investigating arbitrageurs and oracle manipulators in Ethereum. In Proceedings of the 3rd Workshop on Cyber-Security Arms Race (pp. 25-35).

## PREPRINT

4. Amalan, A., **Wang, R.**, Qiao, Y., Panaousis, E. and Liang, K., 2022. MULTI-FLGANs: Multi-Distributed Adversarial Networks for Non-IID distribution. arXiv preprint arXiv:2206.12178.

3. Pejic, I., **Wang, R.** and Liang, K., 2022. Effect of Homomorphic Encryption on the Performance of Training Federated Learning Generative Adversarial Networks. arXiv preprint arXiv:2207.00263.

2. **Wang, R.**, Wang, X., Chen, H., Picek, S., Liu, Z. and Liang, K., 2022. Brief but powerful: Byzantine-robust and privacy-preserving federated learning via model segmentation and secure clustering. arXiv preprint arXiv:2208.10161.

1. Qiao, Y., Chen, C., **Wang, R.** and Liang, K., 2023. FTA: Stealthy and Robust Backdoor Attack with Flexible Trigger on Federated Learning. arXiv preprint arXiv:2309.00127.