ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

RESEARCH GROUP FOR
Distributed Systems

Master Thesis

Institute for Pervasive Computing, Department of Computer Science, ETH Zurich

Scalable and Robust Privacy-Preserving Federated Learning

by Hidde Lycklama à Nijeholt

Spring 2020

ETH student ID:	16-908-808
E-mail address:	lhidde@student.ethz.ch
Supervisors:	Dipl.-Inf. Lukas Burkhalter
	Dr. Anwar Hithnawi
	Prof. Dr. Friedemann Mattern
Date of submission:	June 23, 2020

Abstract

Machine learning algorithms continue to achieve remarkable success in a wide range of applications. These advancements are possible, in part, due to the availability of large domain-specific datasets, for training machine learning models. Hence, there are expanding efforts to collect more representative data to train models for new applications. This raises serious concerns regarding the privacy and security of the collected data. The privacy ramifications of massive data collection in the machine learning landscape have led both industry and academia to work on alternative privacy-preserving technologies for machine learning. Federated Learning is one such promising machine learning technologies that advocates for a new decentralized learning paradigm that decouples data from model training, thus allowing users to retain data sovereignty. However, the large-scale and decentralized nature of federated learning opens it to a new set of privacy and integrity challenges. Ahead of deploying federated learning in high-stakes, privacy-sensitive applications, we must first understand its security and privacy.

This thesis sheds light on the privacy and integrity implications of federated learning. The first part of this thesis consists of an empirical study that analyzes the impact of integrity attacks on federated learning under different strategies, datasets, network sizes, and participation rates. We use the insights from the empirical study to design a new secure federated learning aggregation protocol that provides stronger integrity guarantees. In its essence, the protocol restricts malicious client updates by bounding the norm on top of secure aggregation using homomorphic commitment schemes and zero-knowledge proofs. The use of asymmetric encryption and zero-knowledge proofs introduces a significant overhead that makes the protocol costly to use. To overcome this challenge, we explore and incorporate several optimization techniques from machine learning and compression to make the protocol applicable in practice. Finally, we perform an end-to-end evaluation of the prototype implementation of the protocol. Our evaluation demonstrates that our optimizations reduce the time per round needed from 802 seconds down to 120 seconds (6.82x) for a deep-learning model of 62006 parameters, at a small cost of 0.0306 accuracy reduction. Hence, this first performance assessment shows that our protocol has significant potential for addressing practical security and privacy challenges in federated deep-learning models.

Acknowledgements

I feel incredibly privileged to get the opportunity to work on a purposeful topic together with enthusiastic and intelligent people, without whom this thesis would not have been possible. I want to thank my supervisors, Lukas Burkhalter and Dr. Anwar Hithnawi, for their helpful discussions, motivating advice, and constructive feedback. Their involvement and style of supervising has pushed me to go a step further in pursuing my ideas, and made working together very enjoyable. Also, I would like to thank Prof. Dr. Friedemann Mattern for the opportunity to conduct the master thesis in his group.

Furthermore, I would not have been able to do this thesis without my friends' and family's support. I am very grateful to Nada Abdelwahab for all her love, support, and motivation, which has made this thesis all the more gratifying. Additionally, I would like to thank Joris Oudejans for the many shared endeavors into academics and entrepreneurship that ultimately lead me to complete my studies in Zürich. Last but not least, I want to thank my parents for their everlasting support and for providing an environment in which I am enabled and encouraged to pursue any ambition.

Contents

1 Introduction

Machine learning algorithms continue to achieve remarkable results in a wide range of applications. These advances are enabled, in part, through an increase in the availability of large datasets for training machine learning models. After realizing the benefit that can be gained from data, efforts to centrally collect data have steeply increased. However, this practice of large-scale data collection has raised severe privacy and security concerns. Furthermore, the regular occurrence of unauthorized sharing, selling and use of data, and data breaches, show that the current paradigm of data centralization is untenable [33, 45, 46, 54]. The work in this thesis addresses the tension between user privacy and data utility, by providing a technique to extract advanced utility from decentralized data under rigorous privacy and security guarantees. Doing so contributes to the formation of a new set of essential applications in crucial domains, by enabling accessibility of private data that is currently unavailable due to confidentiality and privacy constraints.

The continuous abuse of user data has elevated the interest in alternative privacy-preserving machine learning methodologies which aim at deriving utility from data without compromising users' privacy. Federated Learning [81] has emerged as a promising machine learning approach that advocates a distributed computing paradigm that decouples the ability to do machine learning from the need to store the data in the cloud. In federated learning, a shared model is trained while privacy-sensitive training data stays decentralized at the client and only small updates specifically intended to train the model are sent to the central provider.

Although federated learning offers a promising solution for data privacy, its decentralized nature opens it up to a host of integrity and privacy problems. Recent work has shown that malicious clients can introduce backdoors into the shared global model by sending malicious updates to the aggregation process [12, 19, 82, 102]. Defenses proposed in prior work, which rely on the statistical inspection [48, 99] or on the use of Byzantine-robust aggregation algorithms [21, 31, 83, 89, 108], have been shown to be circumventable [13]. Moreover, these defenses are incompatible with secure aggregation, which, by design, prevents the inspection of individual client updates; which prior work assumes is feasible by the central provider to protect against information inference attacks [47, 82, 85].

Integrity attacks on federated learning are particularly challenging compared to similar challenges in distributed learning. In this field, a range of Byzantine-robust distributed learning algorithms have been proposed [10, 11, 35, 40], which guarantee convergence under the presence of Byzantine clients by relying on statistics of individual client updates. In contrast, in federated learning, we are confronted with an adversary performing a model poisoning attack that is harder to restrain because it allows the model to converge but with a backdoor. Furthermore, it is impossible to inspect individual client updates because they are inaccessible due to secure aggregation. The focus of this thesis is then to develop a new secure aggregation protocol for federated learning that is robust against integrity attacks from malicious clients.

The research area regarding the integrity of federated learning models is not well explored, and there exist many open questions. Existing attacks are challenging to analyze because they are performed under different attacker goals, training strategies, and environments. Conversely, little work has been done on applicable defenses against these attacks and how these behave for different setups with varying properties such as the attack strategy, dataset, network size, participation rate, learning rate, and regularization.

In this thesis, we provide a thorough analysis of the integrity of models in federated learning to develop a better understanding of this space. We classify and study the impact of existing attacks and propose effective measures to restrict model poisoning attacks by bounding the norm of client updates. To ensure compatibility with secure aggregation, we propose cryptographic proofs to enforce constraints on the norm bound without reducing client update privacy. However, the straightforward use of zero-knowledge proofs comes at a high cost; therefore requires careful consideration of the performance overhead. To this end, we propose several methods to improve the scalability to make our robust, secure aggregation protocol applicable in practice.

1.1 Approach

This thesis presents a new secure aggregation protocol for federated learning that ensures privacy and integrity. The protocol allows federated learning systems to be deployed in large-scale and untrusted environments while preserving privacy and model integrity. To achieve the privacy of client updates, we use homomorphic commitment schemes to aggregate individual client updates, only revealing the aggregate to the central provider. The design of this secure aggregation protocol allows the integration of zero-knowledge proofs, which in our protocol are used to enforce norm bounds on client updates to protect the model against model poisoning attacks from malicious clients. Furthermore, we apply several optimization techniques to make the protocol applicable to large-scale deployments.

1.2 Contributions

More concretely, the contributions of this thesis are:

- The development of an extensive framework that supports the simulation of integrity attacks and defenses, as well as scalability optimizations under a wide range of configurations, models, and datasets, to analyze adversarial federated learning setups;

- Conduct a thorough empirical analysis to understand the impact of integrity attacks on federated learning;
- The design of a new secure and robust federated learning protocol that efficiently incorporates zero-knowledge proofs to enforce norm bounds to defend against model poisoning attacks;
- A prototype implementation of the secure and robust federated learning protocol;
- The proposal and implementation of several performance optimizations to the secure and robust federated learning protocol to improve the scalability of the protocol;
- A comprehensive evaluation of the federated learning system to quantify the performance of our optimized secure and robust federated learning protocol in terms of computation time, bandwidth, and accuracy, against an insecure federated learning baseline and an unoptimized secure federated learning baseline.

1.3 Outline

In Chapter 4, we provide an extensive analysis of integrity attacks and defenses on federated learning and identify norm bounding as a suitable defense mechanism. With the insights from the analysis, we outline the design of a privacy-preserving aggregation protocol that can enforce these norm bounds in Chapter 5. In addition, we propose performance optimizations that improve the scalability of our protocol. Chapter 6 provides an overview of a prototype implementation of the protocol. After that, we perform an evaluation of the implementation in Chapter 7. Finally, we conclude the thesis in Chapter 8 by summarizing the thesis work, describing the potential impact of the project, highlighting open challenges, and advising directions for future work.

2 Background

In this chapter, we discuss the background knowledge that is relevant to understanding this thesis. This thesis lies at the intersection of the machine learning and security domains; hence, this chapter is structured in two parts. In the first part, we discuss background material on machine learning, distributed learning, and federated learning. In the second part, we cover the necessary cryptographic primitives, secure aggregation, and zero-knowledge proofs.

2.1 Machine Learning

Machine learning is the field of creating algorithms to perform a task without explicitly programming the functionality of the algorithm. Machine learning algorithms are used for an increasingly wide range of applications where traditional algorithms would be infeasible to use due to the complexity of the problem at hand. For example, machine learning algorithms are currently used for applications such as spam filtering [8, 38] and self-driving cars [2, 106]. In this section, we start by giving a high-level overview of machine learning and then delve into the underlying core algorithms.

To illustrate the concept of machine learning, let us regard the following example that is used throughout the section. We would like to solve the task of classifying images of objects and animals, such as boats, cars, cats, and dogs, in the correct category. The problem, also referred to as the *task*, is to determine to which class a given image belongs. We assume in this case that there is a fixed number of output classes, in our example 10. An illustration is given in Figure 2.1. This example problem is a fairly complicated task that would take a considerable amount of time to solve using an algorithm that could cover all the different possible input images. Additionally, if we want to write an algorithm for this task, there is a more philosophical question that we must explicitly answer, namely, what exactly determines the image of a cat? While we all have an intuitive definition of what constitutes an image of a cat, it would be very difficult to come up with an exact mathematical description.

Therefore, instead of relying on a crafted algorithm to solve this task, we can resort to a data-driven approach to solve this task; in other words, train a machine learning model for this purpose. A machine learning model is a set of tunable variables that represent a mapping f from the input distribution to the required output distribution. We try to find the right mapping f by identifying the features in the training data that define this mapping, which is also referred to as learning. For this

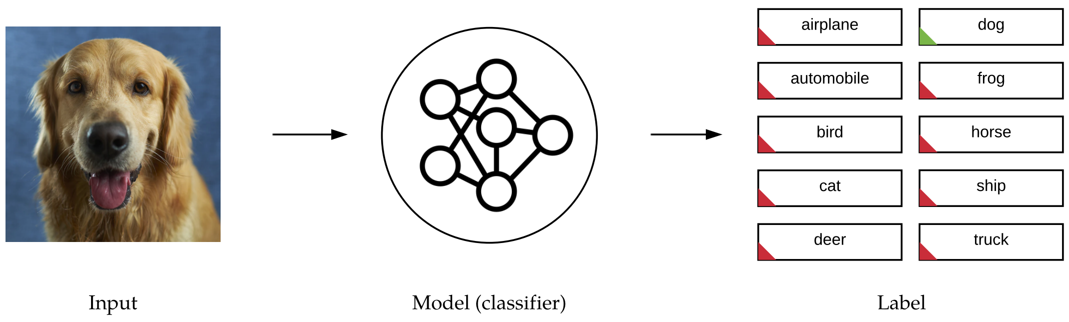


Figure 2.1: The CIFAR-10 image classification task. The model takes a 32x32 image as input and gives a label out of 10 classes as output.

learning we use a task dataset, consisting of tuples of an input x with corresponding label y, is often split into two groups, training data $D = \{(x_i, y_i)\}_{i=1}^{n}$ and validation data $D' = \{(x'_i, y'_i)\}_{i=1}^{n'}$. In our example, the dataset consists of 32×32 pixel images of the various objects and animals, labeled with the corresponding class.

Generally, machine learning models are used in two phases. The previously mentioned learning is referred to as the *training* phase, whereas applying the trained model to make predictions on new data is done in the *inference* phase. During training, the model tries to learn important features from training data using a *learning algorithm*. Learning algorithms can be classified as supervised learning, unsupervised learning, or a combination thereof, such as semi-supervised learning, depending on the presence and quality of the labels for the training data. We use supervised learning in our image classification example because we train the machine learning algorithm on images with corresponding labels.

Model architecture: At a high level, a model can be seen as a network of nodes, organized in layers, as shown in Figure 2.2. Each node represents a computational step and contains references one or more to other nodes from which it receives input. Additionally, every node contains a variable, or *weight*, that is adjustable by the learning algorithm. The model's mapping function f is defined by the weights and biases $\mathbf{w}$. At every node, inputs are multiplied by the node's weight, a bias is added, and a differentiable non-linear function is applied. Given the i-th node x_{ij} with weight a_{ij} and bias b_{ij} at layer j, connected to all nodes in the previous layer $j-1$ of size N_{j-1}, the computational step is defined as follows with activation function σ:

$$x_{ij} = \sigma(\sum_{k=1}^{N_{j-1}} a_{ij} x_{k(j-1)} + b_{ij})$$

The choice of the model architecture together with the non-linear functions allow so-called deep networks to model complex functions. The last layer is referred to

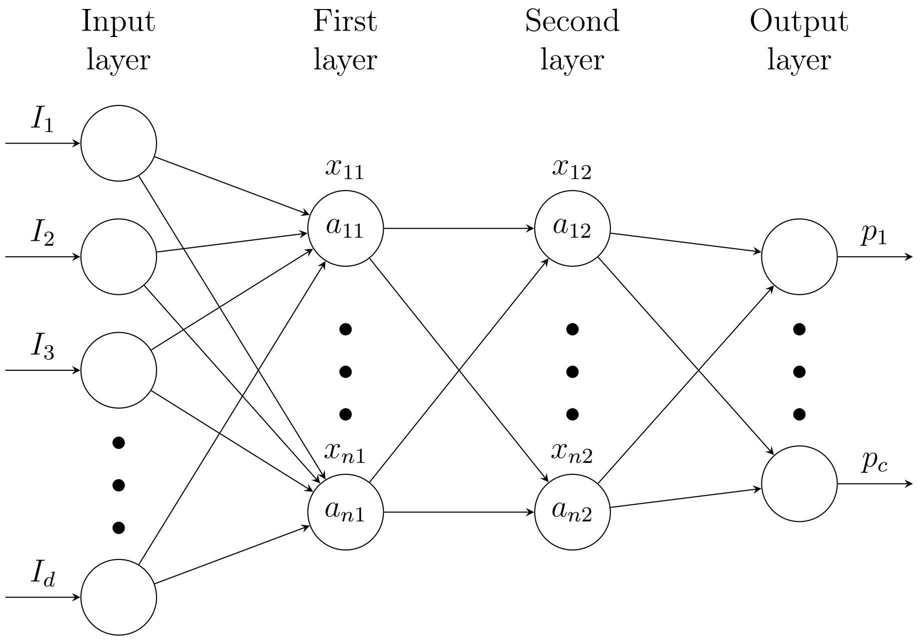


Figure 2.2: Example of a machine learning model architecture for a classification task using fully-connected layers. The network is represented as a directed graph that is organized in layers. The first layer represents the values of the input $I \in \mathbb{R}^d$ given to the network. The nodes in the middle two layers each represent a weight and bias to transform the input with. The final layer represents the probability distribution that the network assigns to each output class. The biases in the nodes in the middle layers are omitted for clarity.

as the output layer and can be seen as the output vector. In a classification task, the output vector is the size of the number of classes C and a softmax function is applied to normalize output vector, resulting in a probability distribution. Every element $p_c \ \forall c \in [0, C)$ in the output vector then represents the confidence of the model of the prediction in that specific class. In our example, this translates to the output layer containing 10 nodes, one for every class. The output class of the model is the index of the weight that contains the highest value.

Loss function: The functionality that we want to achieve is that the model gives us the correct output label for a given input. In the case of our example, the model should give us the correct classification for the object that is depicted in the input image. We define this correct behavior in terms of a loss function, embedded in an optimization problem. Ideally, we would like the network to give us the correct prediction for every class with 100% confidence in that class, and 0% confidence in the other classes. We define this behavior in a loss function ℓ,

$$\ell^{\text{ideal}}(\mathbf{y}, \tilde{\mathbf{y}}) = I(\arg\max(\mathbf{y}), \arg\max(\tilde{\mathbf{y}}))$$

where I is the identity function. The loss function takes the output vector $\mathbf{y}$ as prediction input, as well as the one-hot encoding of the index of the correct label $\tilde{\mathbf{y}}$. The learning algorithm requires every operation in the model to be differentiable, for reasons we will clarify in the next section. Unfortunately, this loss function is not differentiable with regards to $\mathbf{w}$. In order to overcome this, the cross-entropy loss function is often used instead of the identity function:

$$\ell^{\text{xent}}(\mathbf{y}, \tilde{\mathbf{y}}) = \sum_{c=0}^{C} \tilde{\mathbf{y}}_c \log(\mathbf{y}_c)$$

2.1.1 Stochastic Gradient Descent

Learning problems are often formulated as optimization problems with the objective of minimizing the average loss over all samples in the training dataset:

$$\underset{\mathbf{w}}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{N} \ell(f_{\mathbf{w}}(x_i), \tilde{\mathbf{y}}_{\mathbf{i}})$$

For most tasks and networks, the loss function combined with the model is non-convex, resulting in an NP-hard optimization problem. Therefore, a learning algorithm is used to get an approximation of the optimization problem in reasonable time. In supervised learning, the most widely used learning algorithms are based on gradient descent. These algorithms perform optimization in steps based on the gradient of ℓ and f.

Now that we have established what a differentiable model $f_{\mathbf{w}}$ and loss function ℓ are, we define the learning algorithm. From optimization theory, we know that we can find the extremes of a convex function f by taking the gradient and moving along the direction of the gradient, as shown in Figure 2.3. For our model f, we can derive the gradient using the backpropagation algorithm. In backpropagation, the gradient is calculated with respect to each node in the model, moving backwards, starting from the nodes in the output layer and ending at the first layer.

After deriving the gradient for each node for the whole dataset, we update the weights in the nodes. At each training iteration t, the weights of the model are updated with a step η in the opposite direction of the gradient

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta \nabla \ell(\mathbf{w}^{\mathbf{t}})$$

where η is also referred to as the *learning rate*. We take a step in the opposite direction because we are trying to minimize the loss function. Often, the learning rate is decreased with the number of iterations to improve convergence. However, because most widely-used combinations of loss functions and models are not convex, we are not guaranteed to reach a global minimum, but only a local minimum. Nevertheless, in practice, it has been shown that finding a local minimum is sufficient for good performance.

While the gradient descent algorithm works well in theory, it is computationally expensive because, for every step, it requires a pass over the full training dataset, which can be very large. In order to make this more efficient in practice, stochastic gradient descent is used, shown in the Stochastic Gradient Descent Algorithm. In stochastic gradient descent, the gradient is computed for the loss of a randomly selected subset of samples called a minibatch with size b:

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \frac{\eta}{b} \sum_{i=0}^{b} \nabla \ell_i(\mathbf{w}^{\mathbf{t}})$$

The convergence of the model is influenced by the minibatch size b. For instance, using a larger minibatch size may smoothen convergence because the gradient is

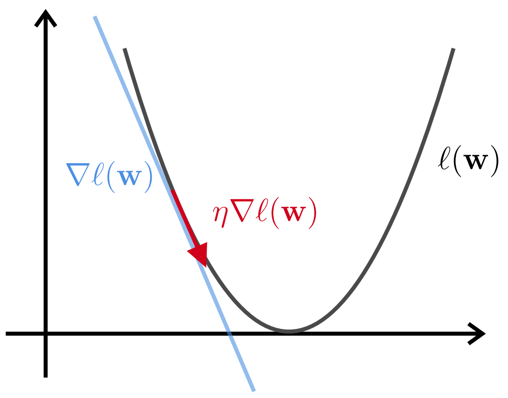


Figure 2.3: Schematic visualization of gradient descent on a loss function ℓ.

computed over the average of b training inputs. Additionally, a larger minibatch size can provide computational improvements up to a certain point due to the potential for parallellization of the operations.

Algorithm 1 Stochastic Gradient Descent

```
1: input: Initial weights $\mathbf{w}_0$, number of iterations $I$, samples $D$, step size $\eta$, loss
   function $\ell$
2: for $i = 1$ to $I$ do
3:     $B_i \subset_R D$                                        ▷ Draw a batch of random samples
4:     $P_i \leftarrow f_{\mathbf{w_i}}(x_{B_i})$                  ▷ Calculate model's predictions
5:     $L_i \leftarrow \ell(P_i, y_{B_i})$                          ▷ Compute loss
6:     $\nabla L_i \leftarrow \frac{\delta L_i}{\delta \mathbf{w_i}}$          ▷ Compute gradient using backpropagation
7:     $\mathbf{w_{i+1}} \leftarrow \mathbf{w_i} + \eta \nabla L_i$        ▷ Update weights
8: end for
9: output $\mathbf{w_i}$
```

2.1.2 Distributed Learning

Machine learning is being applied for increasingly difficult tasks that often require massive datasets and computational capabilities beyond what is supported by a single machine's hardware. Distributed learning was conceived in an attempt to overcome these computational and memory bottlenecks.

In distributed learning, machine learning training is done by multiple worker nodes controlled by a central entity that execute the learning algorithm in parallel in a training *round*. The nodes operate on a shared model that is synchronized after every round. Every node has access to a training dataset, which can be unique per node or be the same dataset for all nodes. A commonly used training algorithm is Distributed Stochastic Gradient Descent (DSGD), this algorithm is a flavour of Stochastic Gradient Descent (SGD) that is optimized for distributed settings.

2.1.3 Federated Learning

Federated learning [81] is a machine learning setting in which a loose federation of clients trains a shared model, orchestrated by a single server while keeping training data decentralized. An important difference between federated learning and distributed learning is the trust model. Distributed learning takes place in a datacenter where the training data and the setup is fully controlled by a single entity. In federated learning, training data is kept local at the clients that act as independent entities, often located outside of the datacenter, as shown in Figure 2.4. Federated learning is particularly suitable for applications where it is problematic to share client data in the clear, i.e., when there are privacy concerns or if there is a bandwidth limitation. We now give two typical examples where federated learning is particularly suitable. One typical example of a federated learning application is to train a language model that predicts the next word as a user types on their phone's keyboard [57]. This setup can involve millions or even billions of phones and is referred to as cross-device federated learning [65]. A second widely used example of federated learning is between multiple organizations wanting to train a shared machine learning model, for example, a consortium of health care providers [86]. Each organization owns a local dataset that it does not want to share with the other participants, but that is too small to train a model on and get meaningful insights from. With federated learning, organizations can collaboratively train a model while keeping their data private, enabling the creation of shared learning that was previously unattainable due to privacy concerns.

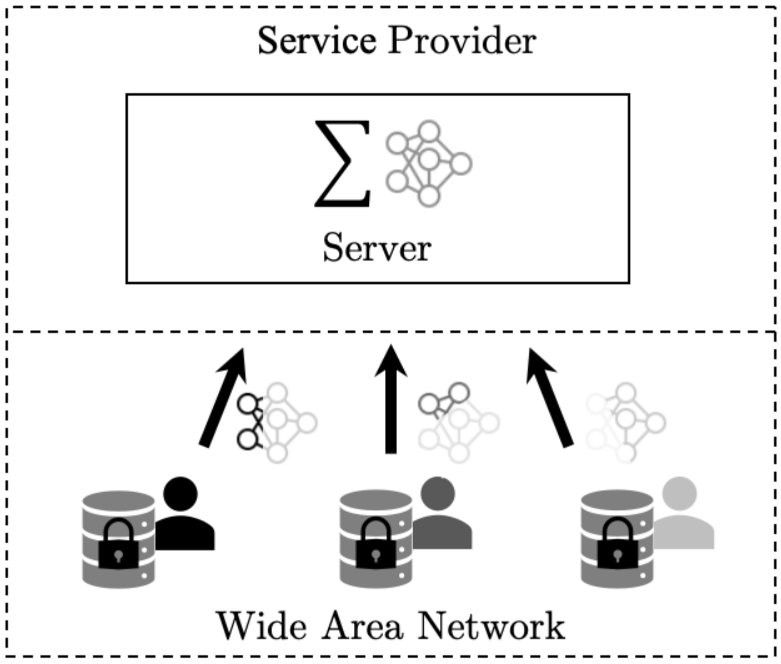


Figure 2.4: Federated learning architecture. Clients submit updates to the central aggregation server, while data stays decentralized.

Federated training algorithm: Training in federated learning is done using the `FederatedAveraging` (`FedAvg`) algorithm. `FedAvg` was introduced as an abstraction of large-batch Distributed Stochastic Gradient Descent (DSGD) and can be configured with several parameters. First, the fraction of clients that participate in a round C. Second, the number of local epochs each client trains E, which can be used to reduce communication costs by aggregating less often. Third, the local batch size B used by the clients. If we set $C = 1, B = \infty, E = 1$, we are left with exactly the large-batch Distributed Stochastic Gradient Descent (DSGD) algorithm from Chen et al. [30]. These parameters can be tuned to find an exact tradeoff between communication and computational costs. For instance, if communication is a bottleneck, it may be useful to increase the number of local epochs each client performs, to increase computation time but to decrease the frequency of aggregation. It has been shown that increasing the number of local epochs is an effective measure and does not reduce accuracy significantly [23].

During training with `FedAvg`, each node draws samples from its local dataset. Ideally, these samples are IID drawn from a local random variable that follows a distribution similar to the global data distribution. However, in the real world, per client samples are often correlated and non-IID over the clients. Nonetheless, it has been empirically shown that algorithms such as `FedAvg` do converge in practice without the IID and convexity assumption. Additionally, it has been shown by Li et al. [77] that the `FedAvg` algorithm converges for a convex objective without requiring the IID assumption.

Weight vector The shape of a model's weights differ per network and the kind of layers that are used. For example, weights in a convolutional layer are organized in a matrix shape, instead of a vector. In the rest of this thesis, for simplicity, we assume a client generates an update in the form a vector $\mathbf{w}$. This vector $\mathbf{w}$ is serialized by the client from the model weights according to the `Flatten Model Weights` algorithm. Upon receiving $\mathbf{w}$, the server deserializes $\mathbf{w}$ to its original weight and layer structure.

2.1.4 Adversarial machine learning

In the last decade, there has been an increasing interest in adversarial machine learning to explore the security risks of machine learning. In this research area, so-called adversarial samples are crafted in an attempt to mislead machine learning models. Insights from this field are relevant to backdoor attacks in federated learning because similar techniques are used. Moreover, the field has developed a theoretical foundation to reason about the robustness against adversarial machine learning, which could be relevant to reason about model poisoning attacks in federated learning. In this section, we briefly describe black- and white-box access and the algorithms used to generate adversarial samples.

Adversarial samples are malicious inputs that look normal to the human eye but are actually carefully crafted to mislead a neural network into giving the wrong output. In our image classification example, given a regular benign input image, an adversary adds small amounts of noise to specific parts of the image to fool the classifier. The adversary has to know exactly what noise to add and can learn this in two ways, depending on its access to the model. Generally, two settings for adversarial model access are used, black-box access and white-box access:

> **Black-box** The adversary can interact with the model by providing inputs and receiving outputs, but can not see what calculations are performed inside the model, or what the parameter values are. A real-world example of black-box access is when the adversary has access to a machine learning model exposed through an API on the internet.

Algorithm 2 FederatedAveraging

```
 1: Input: The $K$ clients are indexed by $k$; $C$ is the fraction of clients selected in
    each round, $B$ is the local minibatch size, $E$ is the number of local epochs, and
    $\theta$ is the learning rate.
 2: Server executes:
 3:     initialize $\mathbf{w}_0$
 4:     for $t = 0$ to $T$ do
 5:         $m \leftarrow \max(C \cdot K, 1)$
 6:         $S_t \leftarrow$ (random set of $m$ clients)
 7:         for each client $k \in S_t$ in parallel do
 8:             $\mathbf{w}_{t+1}^k \leftarrow$ ClientUpdate(k, $\mathbf{w}_t$)
 9:         end for
10:         $\mathbf{w}_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
11:     end for
12:
13: ClientUpdate(k, w):
14:     for each local epoch $i$ from 1 to $E$ do
15:         $w$ = Optimize(w)      ▷ Optimize using training algorithm such as SGD
16:     end for
17:     return $w$ to server
```

Algorithm 3 Flatten Model Weights

```
1: input: Model consisting of layers consisting of weights $\mathcal{M} : \{\mathcal{L}\}$, layer $\mathcal{L} : \mathbb{R}^{n \times m}$
2: $\mathbf{w} = []$
3: for $l$ in $\mathcal{M}$ do
4:     $w = \text{flatten}(l)$                    ▷ Flatten the weights in l
5:     $\mathbf{w} = \text{append}(\mathbf{w}, w)$          ▷ Append to flattened array
6: end for
7: output $\mathbf{w}$
```

White-box The adversary has full access to the model and the parameter values. This means the adversary can perform arbitrary computations on the model. In federated learning, the adversary has white-box access, because every client receives a copy of the global model.

If an adversary has black-box access, there exists statistical techniques to infer what noise should be added to the image [87]. In contrast, with white-box access, creating an adversarial sample is simple. The adversary performs steps of gradient descent on input image i to alter it in order to make the model assign the wrong label. To ensure the adversarial image $\hat{i}$ is not too different from the original image, the adversary defines a constraint for the adversarial image. An example of a constraint is that the adversarial image $\hat{i}$ should be in some ball of radius B around i:

$$\|\hat{i} - i\|_2 \leq B$$

To perform gradient descent under this additional constraint, the Projected Gradient Descent (PGD) algorithm is used. PGD is a generalization of SGD that allows the enforcement of additional constraints on the result set. The PGD algorithm is also used in federated learning settings to craft model poisoning attacks [102].

Algorithm 4 Projected Gradient Descent

```
 1: input: Initial weights $\mathbf{w}_0$, number of iterations $I$, samples $D$, step size $\eta$, loss
    function $\ell$, set of feasible weights under constraint $\mathcal{C}$
 2: for $i = 1$ to $I$ do
 3:     $B_i \subset_R D$                                  ▷ Draw a batch of random samples
 4:     $P_i \leftarrow f_{\mathbf{w_i}}(x_{B_i})$                          ▷ Calculate model's predictions
 5:     $L_i \leftarrow \ell(P_i, y_{B_i})$                                        ▷ Compute loss
 6:     $\nabla L_i \leftarrow \frac{\delta L_i}{\delta \mathbf{w_i}}$              ▷ Compute gradient using backpropagation
 7:     $y_{i+1} \leftarrow \mathbf{w_i} + \eta \nabla L_i$                              ▷ Update weights
 8:     $\mathbf{w_{i+1}} \leftarrow \arg\min_{x \in \mathcal{C}} \|y_{i+1} - x\|$          ▷ Project back onto the feasible set
 9: end for
10: output $\mathbf{w_i}$
```

2.2 Cryptography

In this section, we cover background material on the cryptographic techniques used in this thesis.

2.2.1 Primitives

We start by covering cryptographic primitives that are referenced in this work. For a complete explanation, we refer the reader to [24].

Group homomorphism: Many cryptosystems are defined over finite mathematical structures such as groups or finite fields. Cryptosystems can provide functionality by making use of special properties of some groups. One of these properties is the existence of a group homomorphism between two groups. A group homomorphism is a function in which the group operation between the groups is preserved. An example of this is a partial homomorphic encryption such as ElGamal, that relies on a group homomorphism between $\mathbb{Z}_p$ and a cyclic group $\mathbb{G}$ with prime order p.

Definition 2.1. A group homomorphism is a map $f : G \rightarrow H$ between two groups $(G, \star)$ and $(H, \oplus)$ that preserves the group operation [25], such that

$$f(g_1 \star g_2) = f(g_1) \oplus f(g_2) \qquad \forall g_1, g_2 \in G$$

Pseudorandom generator: Pseudorandom generators (PRG) are used to expand a small random value into a larger random value. They are useful in many applications such as to reduce the amount of bandwidth required when transferring randomly generated values. Instead of transferring the multiple random values, a seed is sent and then expanded into the many values using a PRG.

Definition 2.2. A pseudorandom generator (PRG) is a function G that, given an input seed s from the input space $\mathcal{S}$, produces an output from the output space $\mathcal{R}$. A PRG G is secure if, given an input s, $G(s)$ is computationally indistinguishable from random $r \in \mathcal{R}$.

Typically, the input space $\mathcal{S}$ is a bit-string $\{0,1\}^l$, and $\mathcal{R} = \{0,1\}^L$, where $l \ll L$.

2.2.2 Commitment Scheme

Commitment schemes are an important primitive and is used in many cryptographic protocols, such as zero-knowledge proofs. Intuitively, a commitment scheme can be viewed as a lockbox in which the sender locks a secret value and sends it to the receiver. The content of the box can not be altered by the sender, and not seen by the receiver. The sender can reveal the value of the lockbox by giving the key to the receiver, after which they can open the box and access the secret value. In this section, we explain what a commitment scheme is, and discuss two instantiations of commitment schemes that are relevant to this thesis, namely ElGamal and Pedersen commitments.

A commitment scheme is a cryptographic protocol between a *sender* and a *receiver*. The sender commits to a value or statement. After committing, the sender is unable to change the underlying value of the commitment. The sender is able to reveal the underlying value to the verifier at a later stage, which is also referred to as *opening* the commitment.

Definition 2.3. A commitment scheme is a pair of protocols ($\texttt{Commit}$, $\texttt{Open}$) where the sender inputs x in $\texttt{Commit}$ and the receiver outputs x' after $\texttt{Open}$ [61]. The protocol must satisfy:

– (Correctness.) If the receiver is honest, then $x' \in \{x, \perp\}$
  If both sender and receiver are honest, then $x' = x$.

Additionally, the protocol is secure when the following two properties are met:

– (Binding.) After $\texttt{Commit}$, the value x is fixed.

– (Hiding.) In $\texttt{Commit}$, the receiver does not learn x.

Moreover, a commitment scheme can be:

– (Homomorphic.) $\texttt{Commit}(a+b) = \texttt{Commit}(a) \cdot \texttt{Commit}(b)$

We now discuss two instantiations of commitment schemes.

Pedersen Commitments: The Pedersen commitment scheme is a widely used commitment scheme that is based on the hardness of computing a discrete logarithm in a group [88]. Let $(\mathbb{G}, \star)$ be a group of prime order p and let g and h be two publicly known generators of $\mathbb{G}$. The Pedersen commitment function C_p maps an input $x \in \mathbb{Z}_p$ and a randomly chosen value $r \in \mathbb{Z}_p$ to an element in $\mathbb{G}$:

$$C_p(x, r) \mapsto g^x h^r$$

Pedersen commitments are hiding, because given a commitment z, the receiver can not determine what input x was used to generate z. This is because, for every candidate value x', it is possible to find a value r' that makes the commitment equal to z. We refer to this as *perfectly* hiding, because even if an attacker would have infinite computing capabilities, they could not recover the original x from z, as every x' is equally likely.

Furthermore, Pedersen commitments are not perfectly binding because a prover with infinite computing capabilities could open a commitment z with an arbitrary x' by generating the corresponding r'. However, because finding the discrete logarithm is hard in a group of prime order, we assume there exists an upper bound on the speed with which this can be done. From this follows that Pedersen commitments are computationally binding because it is currently infeasible[1] for a prover to open a given Pedersen commitment to an arbitrary x'.

It is important for the security of the scheme that the prover does not know the discrete logarithm of g with respect to h and vice-versa. This can be ensured by selecting g and h according to a sequence of steps to get verifiable randomness. If the prover knows the discrete log e such that $h = g^e$, they can derive the following equality:

$$g^x h^r = g^x (g^e)^r = g^x g^{er} = g^{x+er}$$

With this, the prover can compute an arbitrary opening (x', r') by finding a pair (x', r') that satisfies the equality:

$$x' + er' = x + er$$

Additionally, the Pedersen commitment scheme is homomorphic, following Definition 2.1:

$$\begin{aligned} C_p(x, r) \star C_p(x', r') &= g^x h^r g^{x'} h^{r'} \\ &= g^{x+x'} h^{r+r'} \\ &= C_p(x + x', r + r') \end{aligned}$$

ElGamal commitments: The ElGamal commitment scheme is another scheme that is relevant to this thesis [41]. Similar to Pedersen commitments, ElGamal

[1]Assuming the group order p is large enough.

commitments operate on a group $(\mathbb{G}, \star)$ of prime order p with independently chosen generators g and h. However, the difference when comparing to Pedersen commitments is that ElGamal commitments contain an additional element of $\mathbb{G}$.

$$C_e(x, r) \mapsto (g^x h^r, g^r)$$

We define the operator $\oplus$ over tuples of elements of G:

$$\begin{aligned} \oplus : \mathbb{G}^2 &\mapsto \mathbb{G}^2 \\ (x_1, y_1) \oplus (x_2, y_2) &= (x_1 \star x_2, y_1 \star y_2) \end{aligned}$$

In an ElGamal commitment, the extra group element g^r binds the commitment to a specific random value r. This alters the binding and hiding properties of ElGamal commitments with regards to Pedersen commitments. First, ElGamal commitments are only computationally hiding, because adversary with unlimited computing power can recover r using a brute-force algorithm. ElGamal thus relies on the hardness of the discrete log for its hiding property. Second, ElGamal commitments are perfectly binding, because the prover only has one way to open an ElGamal commitment, as it is bound to a specific r. Even with unlimited computing power, the prover can not compute a different opening (x', r') that satisfies a given ElGamal commitment.

Furthermore, ElGamal commitments are homomorphic:

$$\begin{aligned} C_e(x, r) \oplus C_e(x', r') &= (g^x h^r, g^r) \oplus (g^{x'} h^{r'}, g^{r'}) \\ &= (g^{x+x'} h^{r+r'}, g^{r+r'}) \\ &= C_e(x + x', r + r') \end{aligned}$$

2.2.3 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange protocol is one of the earliest key exchange protocols that were introduced in asymmetric cryptography. The protocol allows parties to establish a shared secret over an insecure channel, i.e. an adversary can read all messages sent over the channel.

The protocol uses modular exponentiation as a trapdoor function that has some transitive properties. Formally, let G be a group of prime order p with generator g, we define the modular exponentiation function as:

$$\begin{aligned} f &: \mathbb{Z}_p \mapsto G \\ f(x) &= g^x \mod p \end{aligned}$$

The modular exponentiation function is transitive in the exponent: $(g^a)^b = g^{ab} = (g^b)^a$, which is important for the protocol to work. Furthermore, modular exponentiation is a trapdoor function under the assumption the computing the discrete log x for $z = g^x$ is hard.

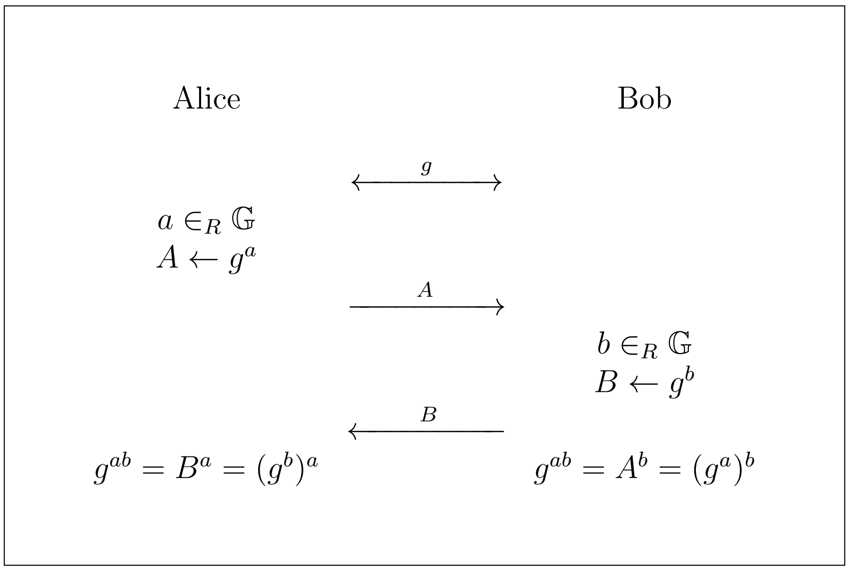


Figure 2.5: The Diffie-Hellman key exchange protocol establishes a shared secret g^{ab} between two parties.

The protocol between two parties Alice and Bob shown in Figure 2.5 works as follows:

1. Alice and Bob agree on a common generator g for the group.
2. Alice chooses a random value $a \in \mathbb{Z}_p$, computes $A = g^a$ and sends A to Bob.
3. Bob chooses a random value $b \in \mathbb{Z}_p$, computes $B = g^B$ and sends B to Bob.
4. Alice computes the shared secret g^{ab} with the message she has received from Bob and her own secret $g^{ab} = B^a = (g^b)^a$.
5. Bob computes the shared secret g^{ab} with the message he has received from Alice and his own secret $g^{ab} = A^b = (g^a)^b$.

Both parties now agree on a shared secret g^{ab}, without it ever being sent over the insecure channel. Concretely, the secret is secure under the assumption that it is infeasible for an attacker to compute a from g^a and b from g^b.

2.2.4 Secure Aggregation

A secure aggregation protocol solves the problem of computing a statistic on multiple private inputs without revealing the individual inputs to any participant, even a central aggregation server. Given a server s and set of n clients with private data $d_1, d_2, d_3, ..., d_n \in \mathcal{D}$, and a function to compute a statistic $\mathrm{F} : \mathcal{D}^n \to \mathcal{O}$ where $\mathcal{O}$ is the set of possible values of the statistic, the goal of a secure aggregation protocol is to compute F while keeping the inputs private and only revealing the output to s.

Secure aggregation protocols are based on different cryptographic primitives, such as pairwise additive masking [9, 52], partially- or fully-homomorphic threshold encryption [71], secret sharing [49], and garbled circuits [14]. Reyzin et al. [93] provide an abstraction of secure aggregation by defining Large-scale One-server Vanishing-participants Efficient MPC (LOVE). Additionally, they formalize the definition of Homomorphic Ad-Hoc Threshold Encryption (HATE) schemes and show how they can be used to build LOVE protocols.

In the context of federated learning, secure aggregation can be applied to protect parties' contributions as these can reveal sensitive information. The first secure aggregation protocol for federated learning was proposed by Bonawitz et al. [23]. In this protocol, secure aggregation is used for clients to submit their privacy-sensitive model updates encrypted such that only the aggregated model is revealed to the server. The protocol works as follows, given a set of clients $\mathcal{U}$ that want to compute the sum over their secret values x_u. Each pair of clients (u, v), $u < v$ agrees on a shared secret $s_{u,v}$ using a key agreement scheme such as Diffie-Hellman. We assume the existence of an arbitrary order relation $<$ on $\mathcal{U}$. Each client u then either adds or subtracts its shared values with each client from x_u, based on the relative ordering of u and v:

$$y_u = x_u + \sum_{v \in \mathcal{U}: u<v} s_{u,v} - \sum_{v \in \mathcal{U}: u>v} s_{v,u}$$

The clients then send their y_u to the aggregator. After receiving all shares, the aggregator computes the sum of the shares. The pairwise shared secrets cancel out, because each secret between a pair of clients $s_{u,v}$ was added to and subtracted from the aggregate once:

$$\begin{aligned} z &= \sum_{u \in \mathcal{U}} y_u \\ &= \sum_{u \in \mathcal{U}} \left(x_u + \sum_{v \in \mathcal{U}: u<v} s_{u,v} - \sum_{v \in \mathcal{U}: u>v} s_{v,u} \right) \\ &= \sum_{u \in \mathcal{U}} x_u \end{aligned} \tag{2.1}$$

For the element-wise aggregation of a vector of values $\mathbf{x}_u$ typical in federated learning, the protocol works in a similar way. In this case, the vectors are aggregated element-wise and the shared secrets are used as input to a Pseudo-Random Generator (PRG) to generate a shared vector of secret values. Moreover, the protocol has additional mechanisms to handle client dropouts and is robust against actively adversarial clients.

2.2.5 Zero-Knowledge Proofs

A zero-knowledge proof is a protocol in which one party proofs to another party of a statement or of knowledge of a value, without revealing any additional information.

Zero-knowledge proofs have many applications due to their privacy-preserving property, for example, in private cryptocurrency transactions [96] or for authentication without transmitting a hash of a password [4]. This section is structured as follows. First, we give an example of a well-known zero-knowledge proof. Then, we formalize the notion of *interactive proofs* and *zero-knowledge.* Next, we go deeper into *proofs of knowledge*, a particular kind of interactive proof. After, we show a method of proving a particular protocol is a *zero-knowledge proof of knowledge* as introduced by Maurer [80].

Schnorr's protocol: In order to provide some context, we first give an example of a zero-knowledge protocol. Given a cyclic group H with generator h and prime order q, this protocol is used to proof the knowledge of a discrete logarithm x of a given $z \in H$, and is known as Schnorr's protocol [97]. With this protocol, a prover can proof knowledge of the discrete logarithm x without revealing x itself. There is no way for the prover to cheat, unless they are able to predict the challenge c that the verifier will send. If the verifier chooses c uniformly at random, the prover has a chance of $\frac{1}{|\mathcal{C}|}$. In order to further reduce this cheating probability, the prover and verifier execute this protocol multiple, say n, times to reduce the probability of cheating to $\frac{1}{|\mathcal{C}|^n}$, which is considered negligible.

$$\begin{array}{ccc} \textsf{Prover} & & \textsf{Verifier} \\ \text{knows } x & & \text{knows } z = h^x \\ & & \\ k \in_R H & & \\ t := h^k & & \\ & \xrightarrow{\quad t \quad} & \\ & & c \in_R \mathcal{C} \subseteq \mathbb{Z}_q \\ & \xleftarrow{\quad c \quad} & \\ r := k + x^c & & \\ & \xrightarrow{\quad r \quad} & \\ & & \text{check } h^r \overset{?}{=} t * z^c \end{array}$$

Figure 2.6: Schnorr's protocol

Interactive proofs: Schnorr's protocol is a proof of the statement that the prover knows a certain x. Furthermore, the proof is interactive because the prover and verifier engage in an exchange of messages in the protocol. This is in contrast to conventional proofs of a statement, which consists of a sequence of verifiable steps that start at one or more axioms and lead to statement to be proved without any interaction between the prover and the verifier. As opposed to a conventional proof, an *interactive proof* is a protocol defined as a protocol between a *prover* P and a *verifier* V. An interactive proof has several motivations over a conventional proof. One motivation is that an interactive proof can be made such that it only transfers the conviction of the statement being true, and does not reveal any additional information.

More formally, an interactive proof is defined as the interaction between a pair of two programs (P, V). An interactive proof must have two properties, *completeness* and *soundness*. Completeness means that an honest prover P will convince an honest verifier V, and soundness means that a dishonest prover P can not convince the verifier V of a false statement. The interaction between P and V creates a so-called *transcript*, denoted with T. T contains all the messages that have been sent between the two parties.

Zero-Knowledge: An interactive proof is *zero-knowledge*, if even a dishonest verifier $\hat{V}$ does not learn anything about the protocol that he did not know before. This is captured by the notion of simulation, first introduced by Goldwasser [51]. $\hat{V}$ could create a transcript T' by simulating the entire protocol by himself, without interacting with P.

Definition 2.4. A protocol (P, V) is *zero-knowledge* if for every efficient program $\hat{V}$ there exists an efficient program S, the simulator, such that the output of S is indistinguishable from a transcript of the protocol execution between P and $\hat{V}$. If the indistinguishability is perfect, i.e., the probability distribution of the simulated and the actual transcript are identical, then the protocol is called perfect zero-knowledge.

Correspondingly, if an interactive proof is zero-knowledge, any transcript T of a real interaction between P and $\hat{V}$ can not be used by $\hat{V}$ to convince another party, because $\hat{V}$ could have generated it himself. This property is called *non-transferability* and holds for every zero-knowledge interactive proof.

Proofs of Knowledge: As stated before, we are interested in *proofs of knowledge*. A proof of knowledge is an interactive proof where the prover tries to convince the verifier of knowing a certain value. Formally, what knowledge is, corresponding to a value z, is defined by the following verification predicate:

$$Q : \{0,1\}^* \times \{0,1\}^* \to \{\texttt{false}, \texttt{true}\}$$

For a value z, the prover claims to know a value x such that $Q(z, x) = \texttt{true}$. The value x is also referred to as a *witness* of z. We will now state the formal definition of a proof of knowledge given by Feige, Fiat, and Shamir [43]. The definition captures the notion that executing the protocol successfully implies the knowledge of a witness x such that $Q(z, x) = \texttt{true}$ [80].

Definition 2.5. An interactive protocol (P, V) is a *proof of knowledge* for predicate Q if the following holds:

- (Completeness.) V accepts when F has as input an x with $Q(z, x) = \texttt{true}$.
- (Soundness.) There is an efficient program K, called *knowledge extractor*, with the following property. For any (possibly dishonest) $\hat{P}$ with non-negligible probability of making V accept, K can interact with $\hat{P}$ and outputs (with overwhelming probability) an x such that $Q(z, x) = \texttt{true}$.

Example 2.2.1. Predicate Q for the knowledge of a discrete log x for y with respect to g:

$$Q((g, y), (x)) = (y \overset{?}{=} g^x)$$

Proving the security of an interactive protocol: Maurer [80] showed that many of the previously found three-step zero-knowledge protocols are actually instantiations of the same protocol. He defines a generic protocol that is a zero-knowledge proof of knowledge to proof the knowledge of a pre-image of a one-way group homomorphism, provided two special properties are met. Conversely, we can treat this as a straightforward method to proof these security properties of any protocol. Similarly, this theory is of interest to our work, because it helps us proof the security properties of our protocol. Therefore, we describe the necessary theorems and steps of this method. We start by introducing the a definition that captures the special property that allows us to argue about the soundness of a proof of knowledge.

Definition 2.6. Consider a predicate Q for a proof of knowledge. A three-move protocol round (prover sends t, verifier sends c, prover sends r) with challenge space $\mathcal{C}$ is *2-extractable* if from any two triples (t, c, r) and (t, c', r') with distinct $c, c' \in \mathcal{C}$ accepted by Vic one can efficiently compute an x with $Q(z, x) = \texttt{true}$

With the following theorem, we can show that the *2-extractability* property of a protocol implies that it is a proof of knowledge.

Theorem 2.2.1. *An interactive protocol consisting of s 2-extractable rounds with challenge space $\mathcal{C}$ is a proof of knowledge for predicate Q if $1/|\mathcal{C}|s$ is negligible.*

Proof. We need to exhibit a knowledge extractor K. It can be defined by the following simple procedure:

1. Choose the randomness for $\hat{P}$.
2. Generate two independent protocol executions between $\hat{P}$ and V. (with the *same* chosen randomness for $\hat{P}$).
3. If V accepts in both executions and the challenge sequences were distinct, then identify the first round with different challenges c and c' (but, of course, the same t). Use *2-extractability* to compute an x, and output it (and stop). Otherwise go back to step 1.

□

If the success probability of $\hat{P}$ is non-negligible, we can show that the expected running time of the knowledge extractor is polynomial.

Now that we have the first theorem towards a proof of knowledge, we have to look at the zero-knowledge property. We need to define another special property of a protocol round, called *c-simulatability*, which is required to construct the zero-knowledge simulator.

Definition 2.7. A three-move protocol round (prover sends t, verifier sends c, prover sends r) with challenge space $\mathcal{C}$ is *c-simulatable* if for any value $c \in \mathcal{C}$ one can efficiently generate a triple (t, c, r) with the same distribution as occurring in the protocol (conditioned on the challenge being c).

Using this definition, we get the following theorem that implies that any protocol with c-simulatable rounds is zero-knowledge.

Theorem 2.2.2. *A protocol consisting of c-simulatable three-move rounds, with a uniformly chosen challenge from a polynomially-bounded (per-round) challenge space $\mathcal{C}$, is perfect zero-knowledge.*

The proof of this theorem is omitted, but we describe the basic idea. To simulate round i the simulator samples c_i uniformly and generates the (t_i, c_i, r_i) triple using the c-simulatability property. The simulator then checks if $\hat{V}$ would actually choose c_i as the challenge in round i. If the check succeeds, the (t_i, c_i, r_i)-triple is appended to the transcript for that round, otherwise the round is started. Note that this assumes that $\hat{V}$ must be rewindable.

Group homomorphisms:

We now formalize the notion of a homomorphism to be able to reference it in our generic protocol. We consider two groups $(G, \star)$ and $(H, \oplus)$ where the group operators $\star$ and $\oplus$ are efficiently computable. Recall from Definition 2.1 that a function $f : G \to H$ is a homomorphism if

$$f(x \star y) = f(x) \oplus f(y)$$

In the next steps, we will assume f to be one-way, i.e. $z = f(x)$ is efficiently computable, but computing x for a given z is infeasible. We do this because only in that case it makes sense for a prover to claim knowledge of a the pre-image x of z where $z = f(x)$. If f would be efficiently invertible, any verifier could get x from z, and a zero-knowledge protocol would be useful for this purpose. However, even if f turns out to be not one-way, the security guarantees of our protocol would still hold.

Generic Protocol

The protocol due Maurer [80] in Figure 2.7 is an abstraction of many previously given proofs of knowledge protocols. The protocol is a proof of knowledge of a value x such that $z = [x]$, for a given z, given that the two conditions in Theorem 2.2.3 are met

$$\begin{array}{ccc}
\textsf{Prover} & & \textsf{Verifier} \\
\text{knows } x & & \text{knows } z = [x] \\
 & & \\
k \in_R G & & \\
t := [k] & & \\
 & \xrightarrow{\quad t \quad} & \\
 & & c \in_R \mathcal{C} \subseteq \mathbb{Z}_p \\
 & \xleftarrow{\quad c \quad} & \\
r := k \star x^c & & \\
 & \xrightarrow{\quad r \quad} & \\
 & & \text{check } [r] \overset{?}{=} t_1^l \oplus z^c
\end{array}$$

Figure 2.7: Generic protocol for proof of knowledge

Theorem 2.2.3. *If values $\ell \in \mathbb{Z}$ and $u \in G$ are known such that*

(1) $\gcd(c_1 - c_2, \ell) = 1$ *for all* $c_1, c_2 \in \mathcal{C}$, *and*
(2) $[u] = z^\ell$,

then the three-move protocol round described in Figure 2.7 is 2-extractable. Moreover, a protocol consisting of s rounds is a proof of knowledge if $1/|\mathcal{C}|^s$ is negligible, and it is zero-knowledge if $|\mathcal{C}|$ is polynomially bounded.

Proof. 2-extractability can be proved as follows: From r and r' such that $[r] = t \oplus z^c$ and $[r'] = t \oplus z^{c'}$ for two different challenges c and c' we can obtain $\tilde{x}$ satisfying $[\tilde{x}] = z$, as

$$\tilde{x} = u^a \star (r^{'-1} \star r)^b,$$

where a and b are computed using Euclid's extended gcd-algorithm such that

$$\ell a + (c - c')b = 1.$$

We make use of

$$[r^{'-1} \star r] = [r^{'-1}] \oplus [r] = z^{-c'} \oplus t^{-1} \oplus t \oplus z^c = z^{-c'} \oplus z^c = z^{c-c'}$$

to see that $[\tilde{x}] = z$:

$$\begin{aligned} \tilde{x}t &= [u^a \star (r'^{-1} \star r)^b \\ &= [u]^a \oplus [r'^{-1} \star r]^b \\ &= (z^\ell)^a \oplus (z^{c-c'})^b \\ &= z^{\ell a + (c-c')b} \\ &= z. \end{aligned} \tag{2.2}$$

Theorem 2.2.1 directly implies that the protocol is a proof of knowledge, and Theorem 2.2.2 implies that is zero-knowledge if $|\mathcal{C}|$ is polynomially bounded since it is c-simulatable. This is easy to see: Given z and a challenge c, one chooses r at random and computes t as $t = [r] \oplus z^{-c}$ □

The resulting Theorem 2.2.3 can be used to proof the zero-knowledge and proof of knowledge properties of any protocol. In order to do this, we only need to specify the groups G and H, the homomorphism f, and check the two conditions of Theorem 2.2.3.

Fiat-Shamir heuristic:

In order to use an interactive protocol in practice, the prover and verifier are required to be online at the same time. This brings a range of additional challenges such as network latency and overhead. Instead, an interactive protocol can be made *non-interactive* using the Fiat-Shamir heuristic [44]. This way, the proof can be generated offline, the proof can be stored and reviewed later by one or multiple verifiers. For instance, this mechanism is used in ZeroCash [16] to store range proofs on a blockchain. ZeroCash is a decentralized digital currency that supports privacy-preserving transactions, which means the currency has the fungibility property. A prerequisite for the Fiat-Shamir heuristic is that the interactive proof is public coin.

Definition 2.8. An interactive protocol (P, V) is called public coin if all messages sent from the verifier to the prover are chosen uniformly at random and independently of the prover's messages, i.e., the challenges correspond to the verifier's randomness ρ.

We can make any public-coin interactive protocol non-interactive in the following way. As an example, we will use the previously seen generic interactive protocol that consists of three communication steps between the prover and verifier. In the second step, the protocol relies on a randomly selected challenge c from the challenge space $\mathcal{C}$ by the verifier. It is easy to see that the protocol is public coin, because the only message sent by the verifier is at the second step, and c is randomly selected independent of t from the first step. In the Fiat-Shamir heuristic, instead of relying on the verifier for this challenge c, a cryptographic hash function is used as a source of randomness for the challenge. The variable t from the first step is used as an input in the hash function.

The Fiat-Shamir heuristic relies on an additional security assumption in order to prove soundness, namely that the output of a cryptographic hash function is sufficiently random given an input. If it were possible for the prover to find a t such that the hash function would output a specific c, the whole scheme would be insecure.

Interestingly, transforming a zero-knowledge interactive proof of knowledge using the Fiat-Shamir heuristic into a non-interactive proof of knowledge removes the zero-knowledge property. This is because the transcript T can now be used by V to convince another party V' because the transcript could not have been generated by V.

2.2.6 Zero-Knowledge Range Proofs

In the previous section, we saw how zero-knowledge proofs can be applied to proof that one has knowledge of a value x. Zero-knowledge proofs can also be used to proof that a statement is true, without revealing any other information. One such statement that is of particular interest to us is that a value lies within a specific range, which we discuss in Section 5.4. In our robust federated learning protocol, range proofs are used to prove additional properties about a client's update, without compromising update privacy. We now formalize the predicate Q that is satisfied in the case of a range proof for a Pedersen commitment.

Given a group $\mathbb{G}$ with generators g and h, let z be a Pedersen commitment $z = g^x h^r$, the property that x lies in a particular range $[A, B]$ is captured by the predicate:

$$Q((x, r), (z, g, h)) = (z \stackrel{?}{=} g^x h^r \wedge x \in [A, B])$$

Much research has been done on range proofs due to their increased applicability in cryptocurrency and privacy-preserving protocols. Specific protocols and instantiations of range proofs are discussed in Section 3.4

3 Related Work

In this chapter, we review research related to scalable and secure federated learning and related work that is relevant to the cryptographic and machine learning techniques we apply in this work. The chapter is structured as follows: We start the chapter by discussing research that focuses on attacks and defenses in federated learning. Next, we review various protocols for secure aggregation. After, we examine techniques to improve the scalability of federated learning. We wrap up the chapter by discussing several implementations of zero-knowledge range proofs.

3.1 Federated Learning

Federated learning can be deployed in the context of machine learning with billions of user devices. In this setting, federated learning provides an improvement over centralized machine learning in terms of bandwidth and data privacy, because training is done at the clients. However, federated learning is inherently more susceptible to integrity attacks than traditional machine learning due to the large-scale distributed organization of the system with many untrusted clients. Some research efforts are focused on studying and analyzing attacks, while others are focused on finding suitable defenses. We cover the previous work in this order.

3.1.1 Attacks

Attacks on federated learning can be categorized as integrity and privacy attacks. We cover both attacks, with more focus on integrity attacks as they are the focus of this thesis.

Research on integrity attacks in federated learning started with Bagdasaryan et al. [12] introducing the *model replacement* attack and showing that federated learning is generically vulnerable to such attacks. Furthermore, they showed that various existing Byzantine-resilient aggregation methods such as Krum [21] can be evaded. Bhagoji et al. [19] proposed a technique for attacking the integrity of a model by specifically targeting situations before the model has fully converged. However, for this they require continuous poisoning over multiple rounds. Sun et al. [102] analyze model poisoning attacks on the Federated-MNIST (FEMNIST) dataset. The FEMNIST dataset was first introduced in the LEAF benchmarking framework for federated learning [27]. They additionally suggest norm bounding and weak differential privacy as effective empirical defense mechanisms against model poisoning attacks. However, some configuration parameters used in this work remain unclear, and it is unclear how the insights drawn from experiments on the FEMNIST dataset can be generalized to other setups. Hence, these unknowns motivated us to further investigate model poisoning attacks. Finally, Tomsett et al. [104] explore similar model poisoning attacks in peer-to-peer distributed learning networks.

The second type of attacks is information inference attacks, aiming to extract sensitive information about training data through the shared global model. There exist many inference attacks where only black-box access to a model is required [59, 78, 100, 105]. However, in federated learning, much stronger attacks exist due to the inherent white-box access to the model [82, 85]. For instance, Melis et al. [82] demonstrate attacks to exploit feature leakage in models. Fredrikson et al. [47] introduce the model inversion attack to construct the average input image for a given class. Information inference attacks are a line of work orthogonal to ours. Defenses against these attacks can be applied complementary to our work on robust privacy-preserving federated learning.

3.1.2 Integrity and Defenses

We can group related work on defenses against model poisoning attacks roughly into two categories, based on the assumptions made in the threat model. The first category of work focuses on model integrity with Byzantine clients. The second category focuses specifically on defending against model poisoning attacks.

Byzantine defenses: Model integrity has been well studied in distributed learning where the concept of Byzantine robustness is used to reason about protocols that operate with potentially faulty worker nodes. A Byzantine client is a worker node that does not conform to the protocol by sending faulty or no messages. A Byzantine-resilient protocol works with the assumption that a fraction of clients is Byzantine. Ideas from these defenses, while more focused on datacenter-like distributed learning setups, may be useful to model poisoning attacks in federated learning.

Several Byzantine-resilient aggregation protocols have been proposed for the federated learning setting. Some algorithms, such as `Krum` and Robust Secure Aggregation rely on robust aggregation statistics like trimmed-mean and the median [21, 89, 108]. These algorithms ensure convergence under the presence of some Byzantine clients. However, as El Mhamdi et al. [40] show, ensuring convergence alone is sufficient for integrity, these solutions do not protect against other poisoning attacks by byzantine clients. Therefore, this is referred to as *weak* byzantine-resiliency, where the algorithms allow for a margin of poisoning. Other Byzantine-resilient algorithms provide strong resilience at an additional cost in terms of resources [35]. Bulyan [40] iterates over some *weak* Gradient Aggregation Rule (GAR) like Krum to provide strong Byzantine-resiliency. Draco [31] computes multiple gradients per client instead of one in an attempt to filter Byzantine clients, which poses significant computational and bandwidth overhead.

Model poisoning defenses: A second, more recent line of defenses attempt to prevent model poisoning attacks. These defenses operate in a setting that is more similar to federated learning where clients are untrusted. This is an arguably stronger threat model than the byzantine setup, because data and update statistics can not be shared freely due to privacy and bandwidth constraints. It has been shown that Krum is not resistant to Sybil and model poisoning attacks in federated learning [48]. As a solution to model poisoning, Fung et al. [48] propose a defense for sybil-based model poisoning using the insight that adversarial model updates are generally more similar than benign updates for non-I.I.D. training data. However, it has also been shown that this defense can be circumvented [13]: Adversaries can decompose their malicious update vector into several orthogonal components that aggregate to the originally intended update. In contrast, the Auror defense relies on the statistical similarity of benign updates [99]. However, the defense has been shown ineffective when the client's dataset is non-IID.

Moreover, all previously discussed defenses, in addition to being ineffective against model poisoning attacks while requiring significant resource overhead, are not possible under secure aggregation. This is because they rely on access to individual client updates, which are protected under secure aggregation.

3.2 Secure Federated Learning

Secure federated learning is a setting where secure aggregation is employed to protect the individual client updates from the server, only revealing the aggregate update. The most widely used protocol is by Bonawitz et al. [23], which was already reviewed in Section 2.2.4. However, this protocol does not provide any protection against actively malicious clients that provide malicious updates in an attempt to attack the integrity of the model. To make matters worse, secure aggregation facilitates this attack, because the individual updates are hidden from the server, removing any accountability of the clients to their update. Other protocols exist that attempt to provide robustness against malicious client updates, which we refer to as robust secure aggregation protocols.

Robust secure aggregation

A protocol with additional robustness guarantees is *Robust Secure Aggregation* by Pillutla et al. [89]. This protocol attempts to aggregate updates based on their geometric median, which is more robust against outliers. They claim that the scheme can be used in complement to any secure aggregation protocol by treating secure aggregation as a black-box component of the protocol. However, the protocol still relies on clients' faithful input in the step where the robust median is computed. Therefore, this step is robust against Byzantine clients but not actively malicious clients. Thus, the protocol is ineffective under a threat model where clients can behave actively adversarial.

The second line of work in robust secure aggregation protocols attempts to solve the problem of robust secure aggregation using multiple non-colluding servers. In large-scale robust secure aggregation, there exists an apparent tension between privacy, robustness, and performance. Corrigan-Gibbs et al. [34] attempt to overcome this tension with Prio, in which a number of non-colluding servers securely compute aggregate statistics. Using secret-shared non-interactive proofs (SNIPs), arbitrary computations can be proved to attest the well-formedness of the input values. Clients share the values to be aggregated along with any SNIPs to the servers. The servers then independently verify the share of the values and the SNIPs by communicating among each other, while preserving privacy. Prio is fast because it relies only on symmetric cryptographic operations. However, this speed comes at a price: the aggregation must be done using at least two servers with the assumption that at least one server does not collude with the other servers. This is a weaker security model, but may be sufficient in practice where two different organizations manage a Prio aggregation server. This method is being used in practice by Mozilla to collect browsing statistics in the Firefox web browser [42, 58]. However, in this case, there still exists the question of where an organization like Firefox would deploy its secondary aggregation server.

3.3 Performance Optimization of Federated Learning

The decentralized nature of federated learning makes bandwidth a crucial performance bottleneck. To address this issue, several techniques have been proposed to reduce the bandwidth cost of federated learning. However, including robustness against malicious clients incurs a significant additional computation and communication cost, which comes with its own set of scalability issues that were previously unaddressed in federated learning. In this section, we review improvements for both problems: Improvements that have been proposed in the context of federated learning without robustness, as well as improvements outside of federated learning that could prove to be useful in our federated learning with robustness setting.

This section is structured as follows: First, we discuss general compression techniques to reduce bandwidth. Then, we look at improvements proposed specifically in the context of federated learning.

3.3.1 Compression

While general compression techniques can significantly reduce bandwidth, they may not always be applicable in combination with the privacy and security constraints of our protocol. We now discuss a compression technique that is generally applicable in signal processing applications, but also relevant in federated learning.

Probabilistic quantization: Quantization is a lossy compression technique to compress a larger, potentially continuous value into a smaller set of discrete quanta. The amount of information required to express values in this smaller set of values is less, providing a reduction in bandwidth. Furthermore, in *probabilistic* quantization, the selection of the quanta is made probabilistically. In the federated learning context, probabilistic quantization is used to compress the amount of information *per parameter* in the client updates. Thus, by using probabilistic quantization, the number of parameters stays the same.

To explain probabilistic quantization, we first describe how it works for compressing an update to a single bit, so-called binary quantization. We then generalize this method to arbitrary n-bit quantization schemes. Let $h_{\min}$ and $h_{\max}$ be the respective minimum and maximum values of the vector that we want to encode. For each element h_j in the update vector, we have a compressed element $\tilde{h}_j$:

$$\tilde{h}_j = \begin{cases} h_{\max}, & \text{with probability } \frac{h_j - h_{\min}}{h_{\max} - h_{\min}} \\ h_{\min}, & \text{with probability } \frac{h_{\max} - h_j}{h_{\max} - h_{\min}} \end{cases} \tag{3.1}$$

The further away h_j is from either $h_{\min}$ and $h_{\max}$, the higher the probability that $\tilde{h}_j$ is the other value. A visualisization is shown in Figure 3.1. Because for every weight only two values are possible, we can encode them using a single bit, i.e., 0 or 1.

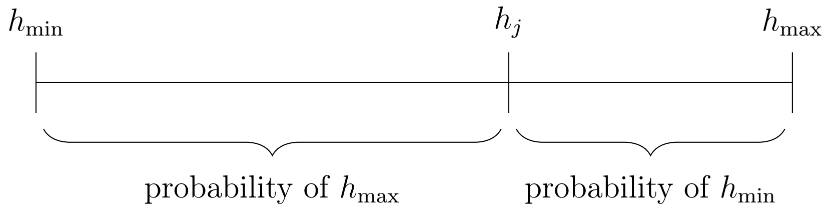


Figure 3.1: A visualization of the probability of probabilistic binarization

Because the values are chosen probabilistically, values of 0 and 1 are selected with a similar frequency that when averaged should lead to a good estimate of the true average h_j. We can proof this by showing that $\tilde{h}_j$ is an unbiased estimator for h_j by using the definitions of $\tilde{h}_j$ and the expected value:

$$
\begin{aligned}
\mathbb{E}[\tilde{h}_j] &= h_{\max} \cdot \mathbb{P}[\tilde{h} = h_{\max}] + h_{\min} \times \mathbb{P}[\tilde{h} = h_{\min}] \\
&= \frac{h_{\max} \times (h_j - h_{\min}) + h_{\min} \times (h_{\max} - h_j)}{h_{\max} - h_{\min}} \\
&= \frac{h_{\max} \times h_j - h_{\max} \times h_{\min} + h_{\min} \times h_{\max} - h_{\min} \times h_j}{h_{\max} - h_{\min}} \\
&= \frac{h_{\max} \times h_j - h_{\min} \times h_j}{h_{\max} - h_{\min}} \\
&= \frac{h_j \times (h_{\max} - h_{\min})}{h_{\max} - h_{\min}} \\
&= h_j
\end{aligned}
$$

If the updates of many clients share the same weight distribution, $\tilde{h}_j$ is likely close to the true h_j. The averaging over many clients cancels out the variance that the binary approximation may cause. In most common machine learning architectures, parameters are usually stored in 32- or 64-bit floating point representations, resulting in a speedup of a factor 32 or 64, respectively.

However, reducing the parameter update to a single bit may lead to a large accuracy loss because of a large approximation error. This can be attributed to the number of clients being too small, the weight distributions of the clients not being similar, or the interval $[h_{\min}, h_{\max}]$ being too large. Fortunately, this scheme can be generalized to reduce parameters to an arbitrary number of bits. This way, the error can be reduced at the cost of extra bits. Instead of binarization, the process is then called quantization. With k bits, we can encode 2^k values, which are chosen by evenly segmenting $[h_{\min}, h_{\max}]$.

To compress a parameter h_j using probabilistic quantization, the process is as follows. First, $h_{j,\min}$ and $h_{j,\max}$ are determined by finding the lower and upper bound in the segment that h_j lies in. Then, $\tilde{h}_j$ is determined using Equation 3.1, similar to binarization. Using probabilistic quantization, we can make a tradeoff between accuracy and bandwidth. It has been empirically shown that probabilistic quantization for client *uploads* is very efficient and barely reduces accuracy on the CIFAR-10 dataset [68]. However, when probabilistic quantization is applied to the client *downloads*, i.e. the global model update sent to the clients, accuracy is severely impacted [69]. This result is in line with our theoretical expectation, as $\tilde{h}_j$ is only a good estimator of h_j if we have multiple samples to average over.

3.3.2 Federated Learning Improvements

We now discuss optimizations that have been proposed specifically in the context of federated learning. Most improvements focus on reducing the number of parameters, which is a bottleneck in training time and bandwidth. Moreover, these optimizations may also prove useful in improving the scalability of additional robustness guarantees. At the same time, the optimizations must be compatible with the robustness guarantees.

Improvements to reduce bandwidth in the context of federated learning focus on reducing the up- and down-link communication cost by compressing the model updates [68, 69]. Konečný et al. [68] categorize these compression techniques into two classes: Structured updates and sketched updates. We organize our discussion of these improvements in a similar fashion.

Sketched updates

Sketched updates are a form of compression where model updates are compressed in a separate step after local training. A client i first creates an update $\mathbf{w}_i$ using a learning algorithm without additional compression constraints and then compresses the update using lossy compression techniques.

Random mask: In this setting, a random subset of entries of $\mathbf{w}_i$, $\tilde{\mathbf{w}}_i$ is transmitted to the server[1]. The server then aggregates the updates $\tilde{\mathbf{w}} = \sum_i \tilde{\mathbf{w}}_i$. A random mask M is generated using a random seed that is generated individually for each client per round. Entries in the random mask are chosen following a Bernoulli distribution with parameter p as the fraction of entries in the random mask that are kept:

$$M_i \sim \mathit{Bernoulli}(p)$$

The random mask is then created from the client's weight update as:

$$\tilde{\mathbf{w}} = (\mathbf{w} \otimes M)\frac{1}{p}$$

where $\otimes$ denotes the Tensor product, i.e. the element-wise multiplication of two matrices. We can now see that the expectation of the aggregated update $\mathbb{E}[\tilde{\mathbf{w}}] = \tilde{\mathbf{w}}$ is an unbiased estimator for $\mathbf{w}$,

$$\begin{aligned}
\mathbb{E}[\tilde{\mathbf{w}}] &= \mathbb{E}[(\mathbf{w} \otimes M)\,\frac{1}{p}] \\
&= \mathbb{E}[\mathbf{w} \otimes M]\frac{1}{p} \\
&= (\mathbf{w} \otimes \mathbb{E}[M])\,\frac{1}{p} \\
&= (\mathbf{w} * p \otimes \mathbb{1})\,\frac{1}{p} \\
&= \mathbf{w} \otimes \mathbb{1} \\
&= \mathbf{w}
\end{aligned}$$

using the linearity of expectation. This compression method works well due to the large amount of clients participating in the network. However, there is a security

[1]This is also referred to as Subsampling [68].

problem with the Random Mask technique, that was not discussed in the original work. Here, as fewer clients contribute to the same parameters, Random Mask can reduce the privacy gained by secure aggregation. For example, the server can generate the random mask in a malicious way such that the sparsity patterns single out a specific client for a subset of the model parameters. In this scenario, the masks are set up so that one client provides the only contribution to some parameters, rendering secure aggregation useless for this subset of model parameters. Furthermore, the client has no way to know if it is the only one participating to this subset of parameters. Alternatively, if the random masks are determined by the clients, a subset of malicious clients may choose to collude and contribute specifically to some parameters of the model. Because only a subset of honest clients will choose to update those parameters, it becomes easier for a subset of adversaries to poison the model. This security issue can be overcome but requires additional steps in the protocol to ensure the sparsity pattern is generated in a verifiable random way.

Structured updates

In contrast to the sketched updates discussed in the previous section, structured updates adapt the optimization algorithm to learn a compressed version of an update directly. For structured updates, the optimization objective is altered to take the compression objective into account.

Random mask: Every client's update $\mathbf{w}_i$ is restricted to be a sparse matrix using some random sparsity pattern that is generated using a seed. The client thus only has to send the non-zero entries of $\mathbf{w}_i$ to the server. This is implemented in a learning algorithm such as SGD by treating the weights that are in the mask as constants, and only computing the gradients for the weights that the client can submit. An advantage of this method over a sketched random mask is that the updates of individual clients can be more meaningful to the global model because no information is lost by zeroing out gradients. Empirically, this method shows promising results in terms of the tradeoff between communication cost and accuracy loss. However, in contrast to a sketched random mask, we do not have any theoretical guarantee that the compressed update is a good estimator for the true update.

Matrix decomposition: A client's update to the local model $\mathbf{w} \in \mathbb{R}^{d_1 \times d_2}$ is decomposed as $\mathbf{w} = \mathrm{AB}$ with $\mathrm{A} \in \mathbb{R}^{d_1 \times k}$ and $\mathrm{B} \in \mathbb{R}^{k \times d_2}$. A is generated fresh per round and per client and is kept constant during training. The client has to find a B such that AB is the best k-rank[2] approximation of $\mathbf{w}$. Note that this does not force the global model $\mathbf{w}_G$ to be of rank k, because different clients create different linearly dependent rows in their individual updates, which are then combined into the global model.

[2] In [68], this is referred to as Low Rank.

The clients construct the matrices A and B as part of the optimization process. Instead of first learning $\mathbf{w}$ and then decomposing it into A and B, B is learned directly by incorporating A and B directly into the computational graph and optimizing with respect to B. A can be generated using a random seed, requiring the clients to only send B to the server. This approach saves a factor d_1/k in communication costs.

Federated dropout: A third technique that has been proposed in [27] is federated dropout, which is inspired by dropout layers used in machine learning [101]. Dropout is traditionally used as a regularization method to reduce models from overfitting on training data. At each training pass, a fraction α of weights in the model are randomly zeroed out. As a result, different sub-models of the original model are trained, causing the full model to generalize more, often resulting in higher accuracy on the validation dataset.

In federated learning, this dropout technique can be used to save bandwidth and could improve accuracy as an added benefit. To apply federated dropout, the server sends a different sub-model to each client. For every client, the server randomly zeroes out a fraction α of the weights and sends them to the client without the zero weights, by organizing the nodes such that no zeroed out weights are in the layers. This way, the number of parameters used is reduced by a factor $1 - \alpha$. Additionally, the server can perform federated dropout without any changes to the federated learning protocol on the client-side.

3.3.3 Intrinsic Dimension

To ensure robustness, we rely on per-parameter security proofs that are costly to create and verify, where the number of parameters contributes largely to the amount of computation required. Therefore, we explore methods that can reduce the number of parameters in a model. We found that Random Subspace Learning, unrelated to federated learning, proposed in the machine learning field, is particularly suitable an optimization for federated learning.

Random subspace learning was introduced by Li et al. [75], not as a compression method, but as a way to look at the intrinsic *hardness* of a machine learning task. In their work, Li et al. [75] investigate the intrinsic dimension of a machine learning model. We are the first to propose the application of this work in a federated learning setup. In this section, we explain the idea behind random subspace machine learning and show how it can be applied to federated learning.

Random subspace learning is used to find the *intrinsic dimension* of an optimization problem. We introduce the notion of intrinsic dimensionality with a toy example from the work. Let $\theta^D \in \mathbb{R}^D$ be a weight vector in a parameter space of dimension D, and let θ_0^D be a randomly chosen initial parameter vector. Now consider an optimization problem where $D = 1000$ and where θ^D is optimized using the minimum squared error function. The error function is defined such that it requires the first 100 elements of θ^D to sum to 1, the second 100 elements to sum to 2, and so forth until we have covered all the elements of θ^D in 10 groups. We can initialize θ_0^D according to any random distribution and optimize the weights of θ^D until the loss is arbitrarily close to zero.

There are many solutions to this problem. In fact, the solution space lives in a 990-dimensional hyperplane: from any point that has zero cost, there are 990 orthogonal directions one can move and remain at zero cost. We define s as the dimension of the set of possible solutions and the intrinsic dimensionality d_{int} as the codimensionality of the solution set inside $\mathbb{R}^D$:

$$D = d_{\text{int}} + s$$

In our example, the intrinsic dimensionality of our problem and solution vector d_{int} is 10 ($1000 = 10 + 990$), corresponding to the 10 constraints that we have on our parameter vector. Even though the parameter space is large, the number of constraints we need to satisfy is small. In other words, at any point where the loss is zero, we have 990 directions to move in while keeping the loss zero.

The previous example gives us an intuition of intrinsic dimensionality in which we could find the solution algebraïcally. However, we would like to find d_{int} for more complicated problems, for example for problems with a data-dependent loss function such as neural networks. To this end, Li et al. define *Random Subspace Training.*

Random Subspace Training works as follows. In standard optimization, steps are taken along the gradient with respect to θ^D directly in the space of θ^D. To train in a random subspace, θ^D is defined as:

$$\theta^D = \theta_0^D + \mathrm{P}\theta^d$$

where P is a randomly generated $D \times d$ projection matrix and d is a parameter vector in $\mathbb{R}^d$ where generally $d << D$. θ_0^D and P are generated once kept fixed during training.

θ^d is initialized to a vector of all zeroes, so initially $\theta^D = \theta_0^D$. Additionally, θ_0^D is initialized according to any distribution that is beneficial to the neural network. During training, steps of gradient descent are taken in the space θ^d. The columns of P are normalized to unit length so that the step size in θ^d matches the same distance in θ^D. This ensures compatibility with existing initialization methods and optimizer algorithms such as SGD and Adam [66]. Furthermore, if the columns of P are orthogonal, P is an orthonormal matrix, resulting in the preservation of the L_2-norm of θ^d, which is useful for our robustness guarantees, which is explained in Section 5.6.4. For this, the columns of P can be orthogonalized, but we assume them to already be almost orthogonal due to the approximate orthogonality of high dimensional random vectors.

Size of the projection matrix: An issue with the projection matrix $\mathrm{P} \in \mathbb{R}^{D \times d}$ is that it can become too large to store in memory for a reasonably sized model. Additionally, performing a dense matrix multiplication between P and θ^d is of complexity $O(Dd)$, which means performing the multiplication will grow linearly in D and d and will thus take a significant amount computation time for large D

and d. In order to reduce the memory and computational requirements for P, the authors propose two methods.

The first technique is by constraining P to be sparse, which, when randomly generated, are known to have approximately orthonormal columns [76]. With a sparse matrix, we can perform sparse matrix multiplications, which are optimized for sparse matrices and much faster than multiplication algorithms that assume dense matrices. The sparse matrix is generated as follows: Each entry is chosen to be non-zero with probability $\frac{1}{\sqrt{D}}$ and, if non-zero, is either -1 or 1 with equal probability. P then consists of d columns of on average $\sqrt{D}$ values. The average space complexity and average time complexity for matrix multiplication are then $O(\sqrt{D}d)$.

The second technique is based on the Fastfood transform [72]. The approach is based on the insight that the product of a Hadamard matrix and a Gaussian vector behaves similar to a dense Gaussian matrix. In other words, in order to multiply our vector θ^d with a square Gaussian matrix M, we can perform multiplication with multiple smaller matrices by decomposing M as:

$$M = HG\Pi HB$$

where H is a Hadamard matrix, Π is a random permutation matrix, and G is a random diagonal matrix with independent standard normal entries. Multiplication with a Hadamard matrix can be done using the e Fast Walsh-Hadamard Transform in $O(d \log d)$ time, and the other matrices can be multiplied in linear time and space. One requirement is that the dimension of M is a power of two. If d is not a power of two, we can zero-pad θ^d. If $M > d$, we can stack multiple independent samples of M to increase the dimensionality of the output. With these methods, we achieve a time complexity of $O(D \log d)$ and a space complexity of D. The complexity of this technique allows for the optimization of very large models. The authors show that is feasible to optimize very large models such as the Pong Reinforcement Learning task which has 1M parameters.

The authors define d_{int100} as the intrinsic dimension required of a task to achieve 100% of the baseline accuracy. However, when the baseline accuracy is only achievable when the task requires very well-tuned models. Moreover, the d_{int100} performance would vary widely between runs, as the regularization effect of subspace training would randomly give tiny boosts in accuracy. Therefore, the d_{int90} is used to measure the intrinsic dimensionality of the tasks, as it provides a good tradeoff between measurement stability and still good accuracy on a task. Results show that some problems require a higher intrinsic dimension than other problems. Note that a task is formulated as an optimization problem together with a model architecture. For example, the intrinsic dimension of MNIST with a fully-connected layer architecture is 750, whereas when using convolutional layers, the intrinsic dimension is only 290.

This technique, while not intended for compression, may be useful in federated learning. There is a large potential for improvements over normal federated learning because only the parameters of θ^d have to be transmitted and aggregated each round.

For the initialization of P, we only have to communicate the random seed that is used to generate P instead of sending the whole matrix.

3.4 Zero-Knowledge Range Proofs

Zero-Knowledge Proofs are a class of cryptographic proofs that we use in this work to ensure the robustness of federated learning aggregation while preserving privacy. In recent years, many new constructions and optimization techniques have been proposed to increase the efficiency of Zero-Knowledge Proofs. This is largely driven by the increased interest and relevance of Zero-Knowledge proofs in many important applications, such as cryptocurrencies [16], electronic voting (e-voting) [18, 22, 53, 55, 63], and smart metering [94]. In this section, we discuss some techniques to construct zero-knowledge proofs and specifically focus on the applications of these proofs as zero-knowledge range proofs.

Zk-SNARK: Range proofs are used in cryptocurrencies such as Zerocash [16,17] to prove the validity of transactions without revealing any of the transaction details, such as the amount. To this end, Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs) are used. Zk-SNARKs can prove the correctness of arbitrary circuits translated to Quadratic Arithmetic Programs (QAPs). The Succinctness of the proofs signifies that the proof transcript is small relative to the circuit being proved and that it can be verified in a short time. This makes zk-SNARKs particularly suitable in blockchain systems because they require little storage space and they are easily verifiable by a node in the network. For instance, a zk-SNARK proof for a Zerocash transaction is only 288 bytes and can be verified in 6 milliseconds [84, 96]. However, proof generation is slow and requires significant memory. Additionally, zk-SNARKs require a trusted party to generate some variables that are relied upon to guarantee security in the protocol and which has to be trusted to discard the knowledge of the relation between the variables. If the party does not do this, they are able to forge proofs, which means security is compromised.

In an attempt to solve this, Zero-Knowledge Scalable Transparent Arguments of Knowledge (zk-STARKs) were proposed [15]. Zk-STARKs have no requirement for a trusted setup, but the proofs are significantly larger, from 45 to 200 Kilobytes per Zerocash transaction [84].

Bulletproofs: Bulletproofs are a type of zero-knowledge proofs proposed by Bünz et al. [26]. Similar to zk-STARKs, they do not require a trusted setup, while the proof sizes are generally much smaller than for zk-STARKs. Bulletproofs have native support for Pedersen commitments, which lets us use range proofs without requiring the overhead of implementing elliptic curve arithmetic in zero-knowledge circuits. Bulletproofs use a recursive inner-product argument to proof a value lies within a given range $[0, 2^n]$. With these optimizations, a single range proof requires only $2\log(n) + 4$ group elements and 5 elements in $\mathbb{Z}_p$. Furthermore, Bulletproofs support efficient proof aggregation, adding only a $2 \log m$ factor where m is the number of independent range proofs.

Arbitrary ranges: Range proof techniques such as Bulletproofs allow a party to prove a value lies within a given range $[0, u^n)$. However, in some cases, we would like to be able to prove for a more granular range than a power of u, or for negative values. Camenisch et al. [29] provide a reduction to create a range proof for an arbitrary interval $[a, b]$ using only two range proofs. Given a value σ and suppose that $u^{n-1} < b < u^n$, to show that $\sigma \in [a, b]$, it suffices to show that:

$$\sigma \in [a, a + u^n] \wedge \sigma \in [b - u^n, b]$$

The formal proof of this reduction can be found in [29].

4 Analysis of Federated Learning Robustness

Ahead of deploying federated learning in high-stakes, privacy-sensitive applications, we need to understand its robustness in the presence of adversaries. The goal of this chapter is to quantify and empirically analyze the robustness of federated learning against malicious clients. In traditional machine learning, an adversary may attempt to compromise the integrity and privacy of the model but is always limited by black-box access to the model. Conversely, federated learning, besides being susceptible to known security and privacy vulnerabilities of a typical machine learning setting, is exposed to a new surface of attacks, due to its open nature, i.e., white-box access by a multitude of agents, which we need to examine carefully. This has significant implications for the privacy of the clients and the integrity of the final model. While some work examined attacks on federated learning [12,19,82,102,104] which we review in Chapter 3, the interaction between attacks and defenses on federated learning systems in various setups has not been studied extensively.

In this chapter, we study vulnerabilities and attacks on machine learning algorithms that are particularly challenging in federated learning. The chapter is structured as follows: First, we define the exact goals we want to achieve with this analysis. We then describe the framework that we developed and used in the empirical analysis. Next, we establish the setup and methodology for the analysis. Afterwards, we formalize and inspect attacks on federated learning. Finally, we look at the effectiveness of various defenses against these attacks.

4.1 Study Objectives

The main question we aim to answer in this analysis is: **What is (1) the effect of integrity attacks on model robustness and (2) how can we mitigate the impact of such attacks?**
Addressing (1) entails answering several questions on integrity attacks:

1. What integrity attacks exist and how are they performed? (Section 4.4)
2. What is the impact of these integrity attacks on model robustness? (Section 4.5)

We address (2) by inspecting the following:

1. What defenses against integrity attacks exist and how well do they preserve model integrity? (Section 4.6)

2. What defenses can sufficiently mitigates integrity attacks? (Section 4.6.1)

This study aims to understand attacks and defenses in a wide set of settings, considering various model architectures, model sizes, and task types. The answers allow us to better understand the effect of client adversaries which could aid us in designing more robust federated learning protocols.

4.2 Analysis Framework

For the purpose of this and future studies, we developed a framework to analyze adversarial attacks in federated learning. The framework has matured to a comprehensive federated learning tool with support for a wide range of IID and non-IID datasets, models, and configuration options. A differentiating factor of this framework over other federated learning frameworks is its capability of simulating adversarial attacks and defenses, combined with various efficiency improvements for protocol scalability.

Configuration. The framework allows users to define setups in configuration files with support for more than 80 different configuration options. Examples of options are the number of clients, the number selected clients per round, the parameters used in machine learning training, weight regularization and data augmentation. Furthermore, the framework supports the specification of multiple runs under different hyperparameters to efficiently find suitable values.

Adversaries. The framework supports the modeling of a broad set of adversaries with different goals. Many attack strategies for adversarial federated learning are supported such as scaling attacks, contamination attacks and attacking under a norm bound using projected gradient descent. Conversely, the framework includes various defense mechanisms to protect against the attacks.

Improvements. In addition to adversarial federated learning setups, the framework supports many techniques to improve the scalability of federated learning. Examples of such techniques include probabilistic quantization, federated dropout, and random subspace machine learning. These can be used to quantify the effect of the improvements on model accuracy, adversarial success and bandwidth.

Scalability. The framework is optimized to be highly scalable and fast. Setups with thousands of clients are supported, while only taking up the memory of a single model. Moreover, the framework can be configured to use multiple worker nodes in parallel to speed up training at the cost of having more models in memory. In addition to scalability for large setups, model training is also fast. While the framework is highly customizable and contains many different training methods to craft attacks, the main training and evaluation loops are optimized to use Tensorflow's fast graph-based execution system. Furthermore, input data is fed and augmented through

Tensorflow's `Dataset` API, allowing for GPU training that is unconstrained by the I/O pipeline. For instance, one epoch of CIFAR-10 on LeNet5 takes 3 seconds on an Nvidia Tesla K80 GPU.

Analysis. Finally, the framework has the capability to calculate elaborate statistics of benign and malicious client updates. For post-training analysis, simple inspection tools such as Tensorboard [5] are supported. At the same time, experimental data can be exported to machine-interpretable data formats such as csv to be analyzed by other data analysis tools.

Implementation. The framework is built in Python 3.x using Tensorflow 2.x [7] with machine learning operations defined in the Keras interface. Furthermore, some custom Keras layers are implemented using Tensorflow 2.x operations, to experiment with custom training techniques such as random subspace learning. Data is processed and analyzed using the NumPy library.

4.3 Experimental Setup

To perform the experiments, we use the previously introduced Federated Learning Analysis Framework as the federated learning setup. In the experiments, there exist three types of actors. A server, multiple clients, and attackers. The system executes a federated learning protocol to train a global model on a global objective. The goal of the server and the benign clients is to create a global model with the highest accuracy at the end of training. The server uses the `FedAvg` aggregation algorithm.

Every client communicates directly with the server and not with any other client, unless they are compromised by the adversary. Each client possesses a local dataset that they use for local training. Depending on the setup, the datasets are either IID or non-IID.

We assume that the adversary can compromise an up to a fraction α clients which can then freely communicate to craft malicious updates. In our experiments, we assume a single attacker for simplicity. Therefore, the number of clients n is always the inverse of α, i.e. $n = \frac{1}{\alpha}$. This is justified, because the updates that clients send are averaged. For instance for $\alpha = 0.1$, the contribution to the model of two adversarial clients in a setup with with 20 clients would be the same as the contribution of one adversarial client in a setup with 10 clients. Additionally, the defense method that we employ does not rely on the analysis of properties such as cosine similarity, but on update size, which is relative to the number of other clients.

4.3.1 Methodology

We use the following datasets throughout this chapter, along with the configurations shown in Table 4.1.

Federated-MNIST: This dataset consists of the same hand-written digits as in the MNIST [73] dataset, but the samples are grouped by the creator of the digits to model a natural non-IID distribution. In total the dataset contains digits 3383 users of roughly 100 digits each [28].

Fashion-MNIST: This dataset is a drop-in replacement for the MNIST dataset but with 28×28 images of fashion items [107].

CIFAR-10: This is an image dataset consisting of 32×32 images of 10 categories. CIFAR-10 poses a significantly harder classification task than the MNIST-like tasks [70, 75]. The samples are non-IID distributed based on the label using a Dirichlet distribution with $\alpha = 0.9$.

Name	Description	Comments
`C1-FEMNIST`	Federated-MNIST, 30 selected out of 3383 clients	Similar setup as used by Sun et al. [102]
`C2-CIFAR10`	CIFAR-10, 50 selected out of 100 clients	
`C3-FASHION`	Fashion MNIST, 10 selected out of 100 clients	

Table 4.1: Overview of the setups used in this analysis.

Experiment data was inspected using Tensorboard [5] and graphs were created using Matplotlib [62]. Experiments were done using the Leonhard [3]; a computation cluster provided by ETH Zurich.

4.4 Attack Strategy

Federated learning introduces many new client-side attack vectors due to its white-box model access by the clients. Attacks on federated learning can compromise privacy, integrity, or both. An overview of existing attacks on federated learning can be found in Table 4.2. In this study we focus our analysis on attacks targeting integrity.

In a centralized machine learning setup, an adversary can attempt to influence the integrity of the model by submitting malicious training samples. This is referred to in the literature as a *data poisoning* attack. For this attack to be successful, the adversary must be able to submit a number amount of malicious samples to influence the training algorithm to successfully inject the malicious behavior into the model. In federated learning, clients directly submit their updates to the shared model which are computed based on local training data. This allows adversaries to submit a corrupted model and is therefore called *model poisoning.* Model poisoning is a much stronger attack than data poisoning because the adversary has control

Attack	**Description**	**Compromises**
Sample reconstruction	Reconstruct samples that were used as training data by other clients	Privacy
Membership inference	Determine if a specific sample was used during training	Privacy
Class representative inference	Recreate samples that are typical for the training dataset	Privacy
Model poisoning	Corrupt the model to perform a specific unwanted behavior	Integrity

Table 4.2: Overview of client-side attacks on federated learning.

over the full training process such as the training parameters, training data, and even the training algorithm. Data poisoning can be seen as an instance of model poisoning, where the adversary is limited to using the same training parameters as all other clients.

The rest of this section is structured in two parts. First, we establish a taxonomy of the strategy that is used to perform model poisoning attacks and categorize existing attacks in this framework. Then, we empirically analyze the impact of model poisoning on model robustness, considering different attacker objectives, attacker capabilities, number of clients, and datasets.

4.4.1 Model Poisoning

Existing model poisoning attacks all follow a similar method. The main idea of the attack is to send a malicious update to nudge the global model towards the direction of an area of the parameter space where a certain *malicious objective* is satisfied. The update must be large to overpower the contributions of the other clients in aggregation. An illustration is shown in Figure 4.1. We organize this section into two parts. First, we provide a taxonomy of the malicious objective. Second, we lay out an abstraction of the commonly used strategy to execute the model poisoning attack for a given malicious objective.

Malicious objective. A model poisoning attack aims to inject malicious behavior into the shared model. The attacker defines this behavior in a malicious objective. However, this malicious objective encapsulates a diverse set of potential malicious goals. For example, in an image classification task, malicious objectives can range from classify all blue planes as birds, to misclassify all dogs as any other class, or just reduce the overall model accuracy. Malicious objectives vary in specificity and impact, and all have different feasibility rates and attacker requirements. To better

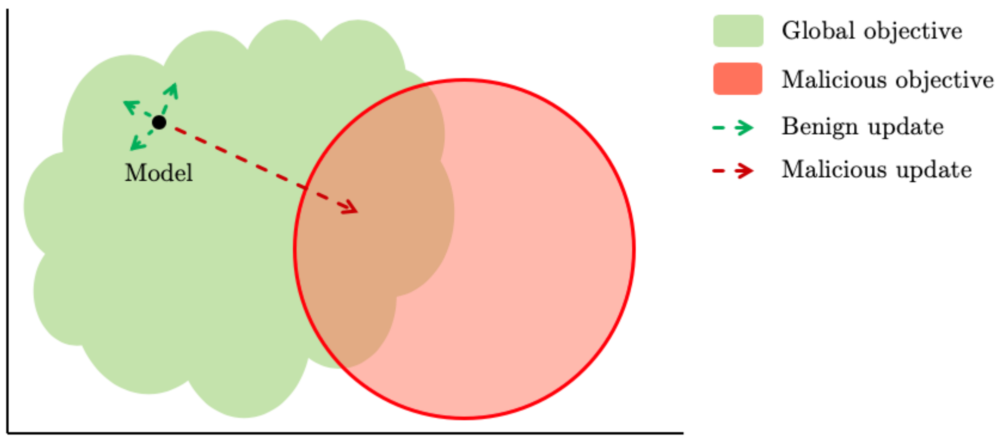


Figure 4.1: Schematic illustration of model poisoning attack. The attacker attempts to modify the model so that it performs a malicious objective while still performing well on the main task.

understand the topic, we provide a categorization of the properties of the malicious objective.

In supervised learning, the malicious objective consists of tuples of a malicious sample and label $\hat{D} = \{(\hat{x}_i, \hat{y}_i)\}_{i=1}^{\hat{n}}$. To optimize the model for the malicious objective, the attacker provides a loss function for the malicious samples:

$$\mathcal{L}_{\text{mal}}(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \mathbf{w}) = \frac{1}{\hat{n}} \sum_{i}^{\hat{n}} \ell_{\text{xent}}(f_{\mathbf{w}}(\hat{x}_i), \hat{y}_i)$$

With a malicious objective, the attacker crafts a malicious update according to a specific strategy. Existing strategies can be generalized as consisting of two steps and an optional third step: Malicious Training, Malicious Scaling, and Global Model Estimation. Before discussing the practical steps, used, we provide a formal definition of the model poisoning attack.

The goal in round t of the attacker, denoted by m, is to make the model in the next round perform well on a malicious objective:

$$\text{minimize}\, \mathcal{L}_{\text{mal}}(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \mathbf{w}^{t+1})$$

However, this optimization is infeasible for the attacker, due to the fact that he does not know the exact weights in round $t+1$. This is because his contribution will be averaged with that of clients that still have to submit their weight updates:

$$\begin{aligned} \mathbf{w}^{t+1} &= \mathbf{w}^t + \theta \sum_{i \in [k]} \mathbf{w}_i^{t+1} \\ &= \mathbf{w}^t + \theta \left(\sum_{i \in [k] \setminus m} \mathbf{w}_i^{t+1} \right) + \theta \mathbf{w}_m^{t+1} \end{aligned}$$

	Description
Digital preprocessing	Whether the attacker preprocesses the samples digitally.
None	Samples are used unaltered
Trigger	A specific trigger pattern is embedded in the training samples. This approach also requires the attacker to have digital access to the testing samples to embed the same trigger and is therefore strictly harder to perform.
Attack precision	What samples the attacker tries to target. It is possible there is some inherent structure in the malicious samples.
Random	Malicious samples are randomly picked, there is no structure between the samples.
Feature	Malicious samples share one or more features.
Byzantine	The most general attack objective where the attacker tries to attack the complete distribution of training samples.
Target class	The classification target of the malicious samples.
Random	Classify the malicious samples as anything but the true label.
Fixed	Classify the malicious samples as a fixed class.
Data distribution	Which parties have access to the malicious samples
Attacker	The attacker controls the malicious samples and the benign clients do not have access.
All	Both the attacker and at least some benign clients have access to the malicious samples.
Type of learning	
Memoization	The attacker wants the model to misclassify the exact training samples.
Generalization	The attacker wants the model to generalize the malicious objective to other, unseen, samples close to the samples defined by the *Attack precision*.

Table 4.3: Taxonomy of the malicious objective

Thus, the attacker has to relax this optimization objective into using an objective that uses the weights in round t or an estimation of the weights in round $t+1$:

$$\text{minimize}\, \mathcal{L}_{\text{mal}}(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \mathbf{w}^t)$$

Malicious Training

First, the attacker trains the model on the malicious objective $\mathcal{L}_{\text{mal}}$ using a method such as stochastic gradient descent. As the attacker has white-box access to the model, the training parameters such as batch size, number of epochs, and learning rate can be completely customized and do not have to be the same as dictated by the server. For instance, it has been shown that quickly decreasing the learning rate can improve the durability of a model poisoning attack throughout multiple rounds [12]. Given the training setup, two kinds of training strategies have been used:

> **Model replacement.** The attacker attempts to fully replace the global model by training on malicious and benign samples at the same time. The attacker draws $b - \hat{b}$ benign samples and $\hat{b}$ malicious samples and puts them in a single batch. The attacker typically performs the optimization for a fixed number of iterations. This method does not guarantee great accuracy on the malicious objective, for instance if the number of iterations is too low and the malicious objective is hard to learn. However, it should allow the model to learn the exact features that differentiate malicious samples from benign samples.
>
> **Segment poisoning.** Instead of having the ambitious goal of replacing the model with an adversarial model, the attacker attempts to poison a part of the model. To do so, the attacker trains its local model on malicious samples, until the loss of the malicious objective is below a target loss L_{targ}. This method guarantees high accuracy on the malicious objective, but may cause the model to have bad performance on the main task.

In addition to the malicious objective, the attacker may use techniques for the update to be close to benign updates to circumvent anomaly detection. The attacker does so by including a second term $\mathcal{L}_{\text{ano}}$ in the loss function $\mathcal{L}$. It has been shown that this approach is successful in circumventing all existing defenses against client-side integrity attacks. The relative weight of both terms in the loss function is regulated by the parameter μ.

$$\mathcal{L}(p, y, \mathbf{w}) = \mu\mathcal{L}_{\text{mal}}(p, y) + (1 - \mu)\mathcal{L}_{\text{ano}}(\mathbf{w})$$

An example of a commonly used $\mathcal{L}_{\text{ano}}$ is a weight regularization term to constrain the size of the weights of the malicious update, by using the L_2-norm:

$$\mathcal{L}_{\text{ano}}(\mathbf{w}) = \|\mathbf{w}\|_2$$

Algorithm 5 Segment poisoning

```
 1: input: Initial weights $\mathbf{w}_0$, malicious samples $\hat{D}$, step size $\eta$, batch size $b$ loss
    function $\mathcal{L}$, malicious loss target $L_{\text{targ}}$
 2: $\hat{L} = \infty$
 3: $i = 0$
 4: while $\hat{L} > L_{\text{targ}}$ do
 5:     $\hat{B} \in_R \hat{D}^b$                    ▷ Draw a batch of random malicious samples
 6:     $P \leftarrow f_{\mathbf{w_i}}(x_{\hat{B}})$             ▷ Calculate model's predictions
 7:     $\hat{L} \leftarrow \mathcal{L}(P, y_{\hat{B}})$                      ▷ Compute loss
 8:     $\nabla\hat{L} \leftarrow \frac{\delta\hat{L}}{\delta\mathbf{w}_i}$       ▷ Compute gradient using backpropagation
 9:     $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i + \eta\nabla\hat{L}$                  ▷ Update weights
10:     $i \leftarrow i + 1$
11: end while
12: output $\mathbf{w_i}$
```

Algorithm 6 Model replacement

```
 1: input: Initial weights $\mathbf{w}_0$, benign samples $D$, malicious samples $\hat{D}$, step size
    $\eta$, loss function $\mathcal{L}$, number of iterations $I$, batch size $b$, malicious samples per
    batch $\hat{b}$
 2: for $i = 1$ to $I$ do
 3:     $B_{\text{ben}} \in_R D^{(b-\hat{b})}$                  ▷ Draw $b - \hat{b}$ benign samples
 4:     $\hat{B} \in_R \hat{D}^{\hat{b}}$                        ▷ Draw $\hat{b}$ malicious samples
 5:     $B \leftarrow B_{\text{ben}} \cup \hat{B}$    ▷ Create batch with both benign and malicious samples
 6:     $P \leftarrow f_{\mathbf{w_i}}(x_B)$                ▷ Calculate model's predictions
 7:     $\hat{L} \leftarrow \mathcal{L}(P, y_B)$                         ▷ Compute loss
 8:     $\nabla\hat{L} \leftarrow \frac{\delta\hat{L}}{\delta\mathbf{w}_i}$       ▷ Compute gradient using backpropagation
 9:     $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i + \eta\nabla\hat{L}$                  ▷ Update weights
10: end for
11: output $\mathbf{w_i}$
```

After getting a poisoned model $\mathbf{w}_m^t$ through malicious training, the attacker creates a malicious update by subtracting the global model $\mathbf{w}_G^{t-1}$ from this model.

$$\Delta\mathbf{x}_m^t = \mathbf{w}_m^t - \mathbf{w}_G^{t-1}$$

Malicious Scaling

After the attacker has created a malicious update that will poison the model, it has to overcome an additional hurdle. Recall from the `FedAvg` algorithm, that the model is averaged over n clients and applied to the global model with a global learning rate of θ. In order for the update to survive the averaging and overpower the learning rate, it is scaled by a scaling factor γ. The scaling factor is set to:

$$\gamma = \frac{n}{\theta}$$

in order for the malicious update to completely survive averaging. However, we show that, in practice, a lower scaling factor can also be sufficient to get moderate adversarial success.

Global Model Estimation

Finally, as an optional last step to improve the success rate, an attacker may choose to negate the updates submitted by the other clients. As the attacker does not have direct access to the clients' updates, it has to estimate what the other clients will send. The attacker, denoted with m can do this by estimating the next average update the other clients $[k]\backslash m$ will send $\Delta\mathbf{w}^t_{[k]\backslash m}$ in a round t by comparing the current global model with the last model it received at a previous round t'.

$$\Delta\mathbf{w}^t_{[k]\backslash m} = \frac{\mathbf{w}^t_G - \mathbf{w}^{t'}_G - \Delta\mathbf{w}^{t'}_m}{t - t'}$$

The attacker then subtracts this estimation from its scaled malicious update.

$$\Delta\mathbf{w}^t_m = \gamma\Delta\mathbf{x}^t_m - \Delta\mathbf{w}^t_{[k]\backslash m}$$

4.4.2 Classification

In this section, we analyze and categorize the attack strategies of model poisoning that is covered in recent literature. The attacks are all instantiations of the attack strategy laid out in the previous section. An overview and classification of the existing model poisoning attacks are shown in Table 4.4.

The table illustrates the many dimensions of the problem space of the analysis. The proposed attacks each have their own focus and characteristics, with different datasets, malicious objectives, malicious training strategies, number of clients, and number of selected clients. In addition to this, the various machine learning hyperparameters such as learning rate and regularization also impact the results of attacks and defenses. This makes the attacks hard to compare.

For some work, not all parameters are described in such a way that the work is reproducible. Attacks W1 and W3 in Table 4.4 contain parameters that were not described in the original paper but that are used in the experiments. In particular, The segment poisoning strategy requires a specific target loss for the malicious samples. Next, W3 does not come with any source code and lacks some descriptions about the methods used for training, specifically for the projected gradient descent method.

	W1: Bhagoji et al. [19]	W2: Bagdasaryan et al. [12]	W3: Sun et al. [102]
Malicious objective			
Digital preprocessing	None	None	None
Sample structure	Random	Feature	Feature
Target class	Fixed	Fixed	Fixed
Data distribution	Attacker	Attacker	All
Type of learning	Memoization	Generalization	Memoization
Setup			
Dataset	Fashion-MNIST	CIFAR-10, Reddit	Federated-MNIST
Malicious dataset size	1 sample	30/16 samples	310 samples
Attack strategy	Segment poisoning	Data poisoning	Data poisoning[a]
Global model estimation	yes	yes	yes
Weight regularization	L_2	L_2	-
Weight constraint	-	-	L_2-ball
Training setup (number of clients/selected per round/malicious)	10/10/1	100/10/1	3383/30/113
Global learning rate	1[b]	1	1
Hidden parameters	yes, LM intensity	yes, DP parameters	yes, PGD method

Table 4.4: Overview of attacks on federated learning.

[a]This is an estimation based on the paper. The authors do not describe the exact methods used for training and no code was published.
[b]A learning rate of 1 replaces the full model in this setting where all 10 clients are selected every round.

4.5 Attack Impact

We now evaluate existing attacks and empirically analyze the impact of model poisoning attacks. The attacks we demonstrate are based on existing attacks in the literature and summarized in Table 4.5.

Attack	Description	Source
C1-FEMNIST		
`A1-HAND`	Classify images of 7 of a subset of handwriters as 1.	[102]
C2-CIFAR10		
`A2-WALL`	Classify images of cars with a specific pattern in the background as birds.	[12]
`A3-GREEN`	Classify images of green cars as birds.	[12]
`A4-STRIPES`	Classify images of cars with a racing stripe as birds.	[12]

Table 4.5: Attacks used throughout this chapter.

The goal is to get an empirical intuition of the behavior of model poisoning attacks under different strategies and in various settings. The rest of this section is organized as follows. We first analyze the impact of model poisoning attacks under different attack strategies and malicious objectives. We then look at the tradeoff between detectability and attack success introduced by stealth methods. Afterwards, we examine the influence on attack success of properties related to the federated learning setup such as the number of clients.

4.5.1 Strategies

We first show the effectiveness of the unconstrained model poisoning attack under different training methods and malicious objectives, to get an intuition on the severity of the attacks.

Single-shot attack. To start, we show the attack in its simplest form. We let the model converge and then have a single attacker perform an attack in one round, using the model replacement malicious training method. As shown in Figure 4.2, the effect is detrimental for model integrity. The model accuracy degrades slightly while the backdoor is injected successfully and stays for many rounds. Additionally, we see the effects of the different malicious objectives. The accuracy of the backdoor is highly dependent on the characteristics of the malicious objective.

Malicious Training. In the previous single-shot attack, we showed the effects of the model replacement training strategy. A comparison with the segment poisoning strategy is shown in Figure 4.3. Both attacks are successful in injecting the backdoor

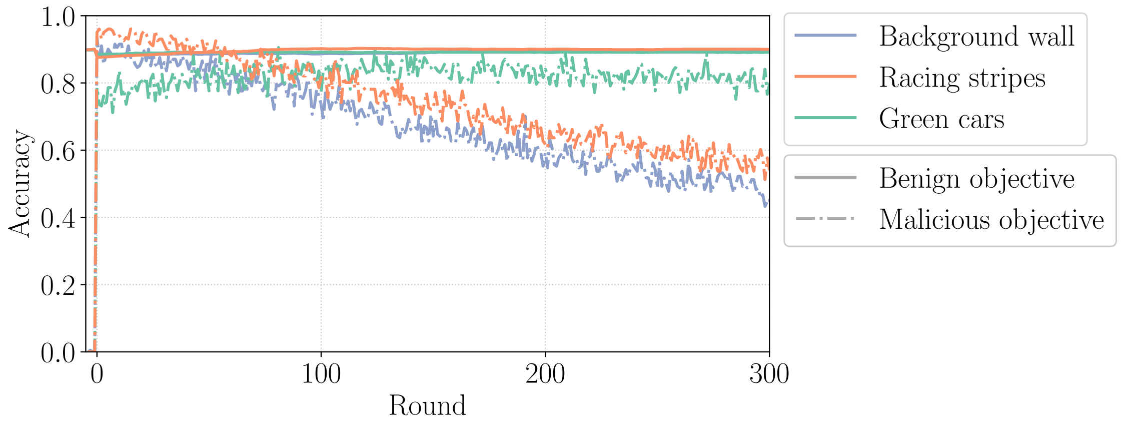


Figure 4.2: Model Poisoning attack. Model: ResNet56. `C2-CIFAR A2-WALL A3-GREEN A4-STRIPES`.

into the model, with an accuracy of around 0.9 for the malicious objective. After poisoning, the backdoor stays in the model for many rounds. However, we see two differences between the two strategies.

First, the model replacement attack keeps a higher accuracy in the model for many more rounds than the segment poisoning attack, which quickly decreases to 0.35 after 75 rounds and then slowly decreases further. The observation that the segment poisoning attack is removed from the model relatively quickly is in line with results reported in [12], which conclude that segment poisoning requires the attacker to participate in every round to maintain backdoor accuracy. Second, the segment poisoning training strategy significantly reduces the global model accuracy, compared to model replacement. We attribute both differences to the fact that segment poisoning only focuses on training the subspace in which the malicious objective is satisfied, whereas model replacement attempts to replace the full model with a model that performs well on both the main task as well as the backdoor task.

While the model replacement attack seems to poison the model for more rounds and affects accuracy on the main task less than the segment poisoning attack, there is a tradeoff in terms of detectability. Specifically, the model replacement attack creates a much larger malicious update when comparing to the segment poisoning attack. The L_2-norm of both updates are 6.53e+4 and 5.28e+4 for model replacement and segment poisoning, respectively. Intuitively, this makes sense because the model replacement attack attempts to replace the full model, whereas the segment poisoning attack replaces only part of the model.

Malicious scaling. An additional component that is very important to model poisoning is the malicious scaling step where the update is scaled to survive averaging using the scaling factor γ. We show the effect of the scaling factor γ on the adversarial success in Figure 4.4. The L_2-norm grows linearly with the scaling factor, whereas

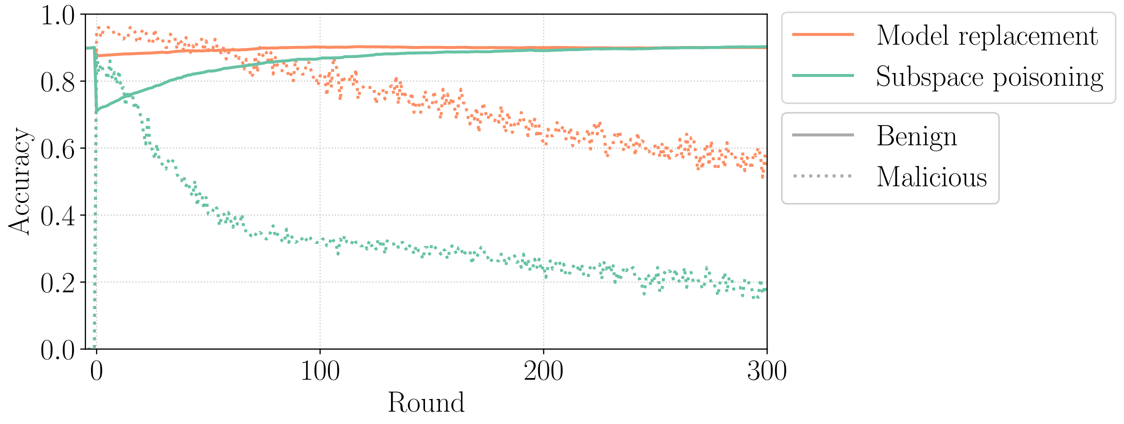


Figure 4.3: Comparison of malicious training strategies. Attack at round 0. `C2-CIFAR A4-STRIPES`.

the success of the attack follows a shape similar to the sigmoid function. In order for an attack to be considered successful, the update must at least be scaled by 20.

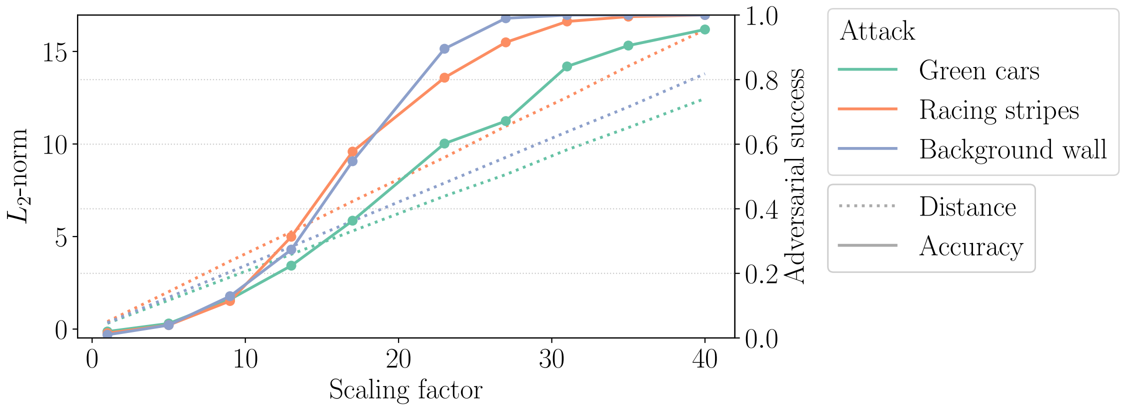


Figure 4.4: Effect of the scaling factor γ on adversarial success and size of the malicious update. $\alpha = 2.5\%$. `C2-CIFAR A2-WALL A3-GREEN A4-STRIPES`.

4.5.2 Environment

Finally, we look at the effectiveness of the model poisoning attack under different numbers of clients. Figure 4.5 shows a setup with under different adversarial fractions α. We see that the number of clients has no effect on the adversarial success. This is because an attacker can simply boost his attack by γ to outweigh any update of benign clients. Therefore, even in a large setup with millions or billions of clients, a single compromised client can poison the model in one round. In the next section, we discuss defenses that aim to prevent this extreme vulnerability.

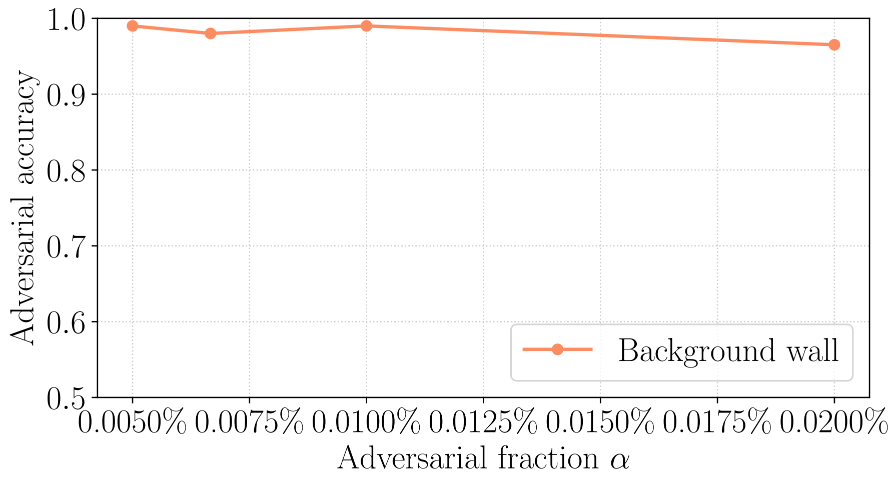


Figure 4.5: Effect of the number of participating clients on the adversarial success of the model poisoning attack. `C2-CIFAR10 A2-WALL`.

4.6 Defenses

In this section, we analyse defenses against model poisoning attacks. In the literature, we have seen two lines of defenses. The first line proposes defenses to ensure Byzantine resiliency, such as Krum, Bulyan, trimmed mean, and coordinate-wise median [21, 40, 108]. The second line focuses specifically on the detection of model poisoning, such as Auror [48, 99]. However, it has been shown that every defense can be circumvented, even without knowledge of the exact defense mechanism being used [13]. Furthermore, all defenses rely on some kind of statistical inspection of the update or rely on the honest self-reporting of update statistics, and are therefore not compatible with secure aggregation.

Thus, there is a need to look for defenses that are compatible with secure aggregation under our threat model. We now present a proposal for an empirical defense based on update size. We start by reviewing the properties of benign and malicious updates and factors that contribute to the size of an update. After, we show that clipping the update by using a norm bound is an effective defense to model poisoning attacks. Finally, we show how to find a suitable norm bound for a given setup.

4.6.1 Update size

In the previous model poisoning attacks, an adversary crafts an update that changes the model such that it performs a malicious objective. In order to do this successfully, the adversary has to craft an update that has an objective that is different from the global model, and that is sufficiently large to overpower the benign clients' updates during aggregation. Both of these requirements contribute to the size of the update of the adversary, shown in Figure 4.6. The L_2-norm of the malicious clients' updates is 8.38 versus 0.99 for benign clients. Moreover, there is a clear difference in the distribution of weight values: Malicious clients' weights lie roughly in the range $[-0.1, 0.1]$, whereas benign client's weights are roughly in the range $[-0.015, 0.015]$ We now further investigate the impact of the update size.

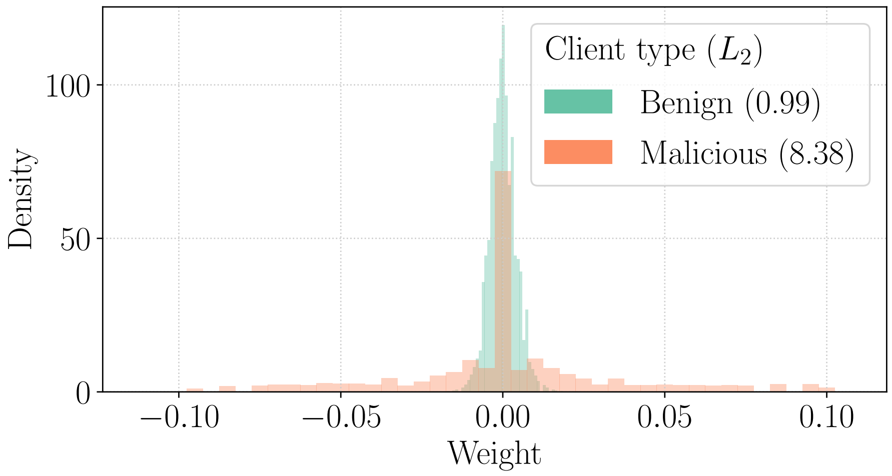


Figure 4.6: Weight distribution and L_2-norm for malicious and benign clients. `A2-WALL`, Segment poisoning, Adv success 1.0, 40 clients.

In each round, every client starts with the same global model and performs local training to create an update. The update characteristics shown in Figure 4.6 raise the question of what exactly influences the size of an update. If this is known, it would be possible to work with a certain expectation of an update. The influence on the size of an update can be attributed to two factors. The first factor is the training configuration, such as the hyperparameters, as well as the concrete model being used. Recall from Section 2.1.1 that each client updates their local model $\mathbf{w}$ using steps of stochastic gradient descent with learning rate η:

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} + \eta \nabla \ell_{\mathbf{w}^{t}}$$

Each client performs r rounds of local gradient descent before sending the update to the server. This r is influenced by the number of epochs e and batch size b, as well as the size S of the local client dataset.

$$r \approx \frac{eS}{b}$$

Clearly, the hyperparameters such as the learning rate, number of epochs and batch size are the same for all honest clients. Furthermore, an estimation can be made about the expected size of a client's dataset.

The second factor is determined by the loss ℓ of the local dataset in relation to the global model. This loss is expressed in the equation as the loss $\nabla\ell$. It can intuitively be understood as how much the local client's dataset agrees with the global model. If the global model has converged and can predict a lot of the samples right, the loss will be small and, hence, the update after one step of gradient descent will also be small. We would therefore expect clients to submit updates with larger norms at the start of training, but the norms will become smaller as the model converges. We can see this effect over time in Figure 4.7. The L_2-norm of the benign clients is predictable and decreases over time. In contrast, poisonous updates are larger because the malicious objective partially contradicts the global objective.

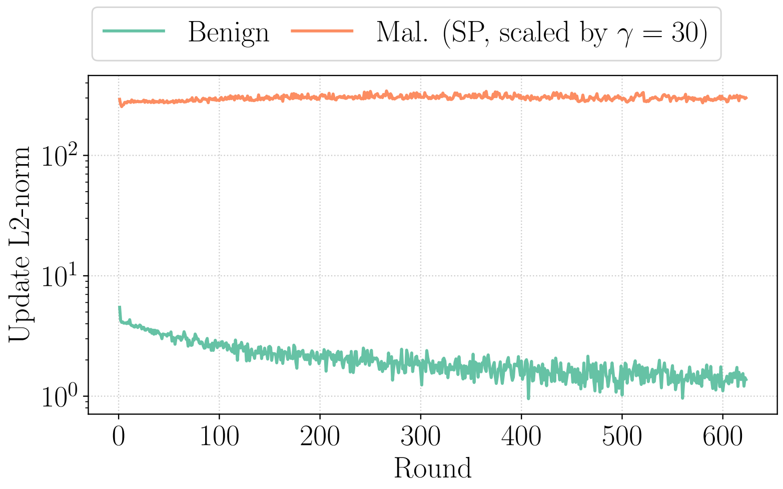


Figure 4.7: L_2-norm of benign and malicious updates over time. In every round, the attacker crafts an update, which is not included in the averaging. `C1-FEMNIST`, 3383 clients, 30 selected.

Moreover, Figure 4.8 shows a quantification of attacks in terms of L_2-norm, compared to the norm distribution of the benign clients. The model poisoning updates increase in size with the adversarial success, and is also inversely correlated to the fraction of malicious clients α. From this we conclude that model poisoning requires a significant update size. Additionally, we can see that some malicious objectives require a larger update size to inject than others. For example, the `A2-WALL` model updates are consistently smaller than that for the other malicious objectives. This is in line with the hypothesis made in [12], namely that the `A3-GREEN` objective is closer to the benign clients' distribution than the `A2-WALL` objective and that this objective is therefore more successful.

4.6.2 Norm bound

In the previous section we showed that update size differs significantly between benign and honest clients. In this section, we show that it is possible to exploit this finding in practice to create a suitable defense. The main idea is that the norm bound blocks the attacker from submitting a large malicious update, but does not interfere with the contributions of benign clients. Figure 4.9 shows clearly the effectiveness of this mechanism: The attacker can not overpower other clients by submitting a large

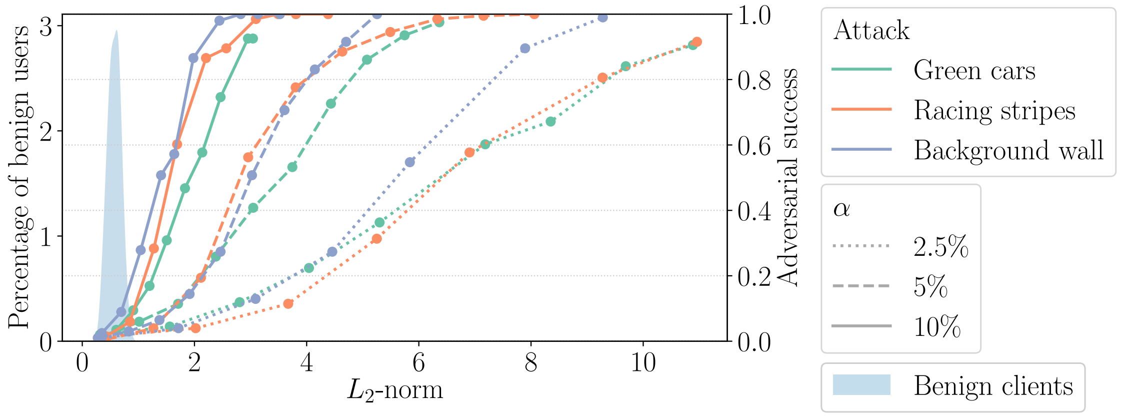


Figure 4.8: Comparison of the L_2-norm of attacks under different participation rates. `C2-CIFAR10 A2-WALL A3-GREEN A4-STRIPES`, Segment poisoning, LeNet5.

update, while other clients stay within the norm bound. This makes the contribution of the benign clients and the malicious clients more proportional to each other.

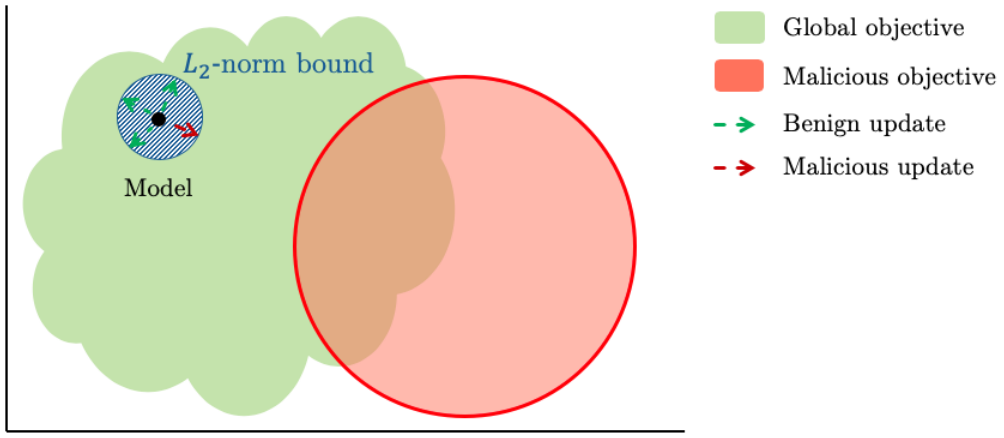


Figure 4.9: Visualization of a federated learning model with a L_2-norm bound for the client updates.

We now look at the empirical performance of a setup where the server enforces a norm bound. In this scenario, the model is under constant attack by the adversary with a frequency of every other round. Figure 4.10 shows this setup in three experiments: A baseline where benign clients can train the model freely, an experiment where updates are clipped, and an experiment with an active adversary together with the clipping bound. Comparing the baseline to the experiment with the active adversary, we see that there is no decrease in accuracy. Additionally, the accuracy for the malicious objective stays low. From this we conclude that the norm bound is an effective defense against this attacker.

Furthermore, from the comparison of the baseline with the clipped experiment, we see that the clipping on its own does not decrease accuracy. Interestingly, we see that the attacked experiment and the clipped experiment actually achieve a higher accuracy for the benign objective. We conjecture that the L_2-norm clipping achieves some kind of regularization effect, which is beneficial for the non-IID federated learning setup.

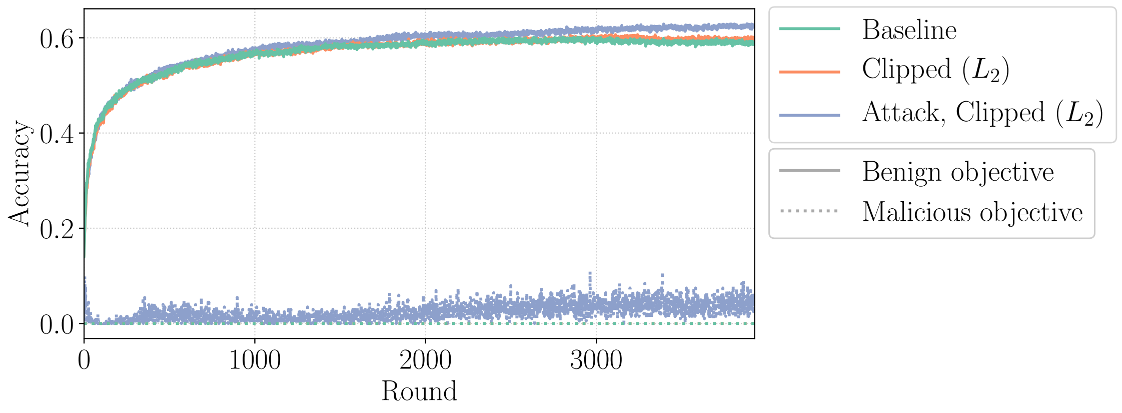


Figure 4.10: Performance of a network with L_2-norm bound 0.8. `C2-CIFAR10`, Subspace poisoningattack, LeNet5, 100 clients, 50 selected per round.

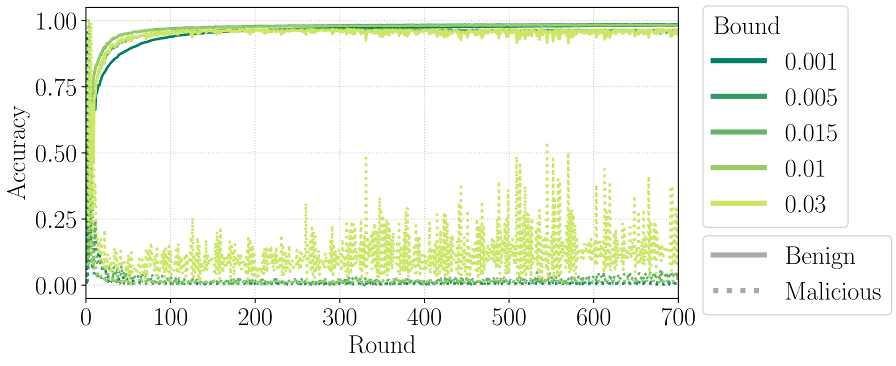
Norm effect

The results shown in the previous paragraph suggests that a norm bound can limit adversarial success while not disturbing model accuracy. We now further investigate the effect of the norm bound on the model. Figure 4.11 shows the performance of the global and malicious objectives under the L_2 and L_∞ norm bounds. At either extreme of the norm range, the accuracy of the global objective diminishes. If the norm bound is too low, benign clients can not train the model enough and the accuracy stays low. Conversely, if the norm is too high, adversaries can get too much influence and poison the model, resulting in a high adversarial success and a lower model accuracy on the main task.

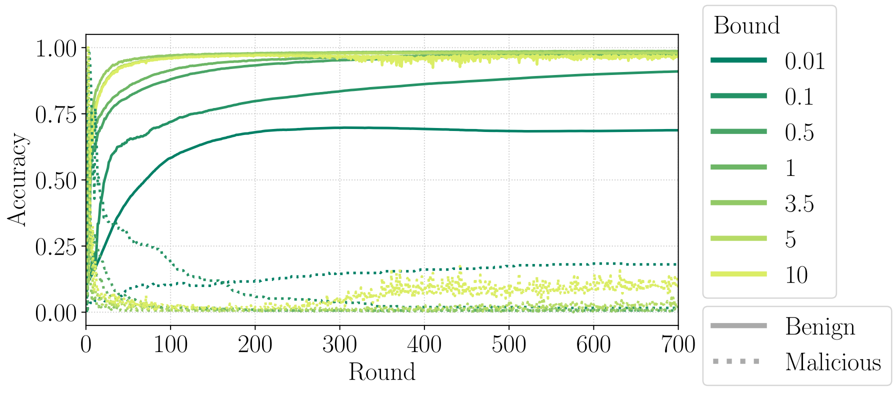
Figure 4.12 provides further evidence for this point, by displaying the accuracy on the main task under various L_∞- and L_2-norm bounds. The results suggest there exists an optimal point somewhere in the middle where the accuracy is close to the baseline, i.e. the setting without any attacker or norm bound. At this particular point, the attacker is not able to sufficiently scale the malicious update to inject the backdoor, while benign clients can still provide enough influence.

Comparison of both norms

Figure 4.12 shows that the L_∞-norm affects global model accuracy significantly after a bound of 0.00015. In comparison, this is a harsh effect in comparison to the



(a) L_∞-norm



(b) L_2-norm

Figure 4.11: Performance of the continuous model poisoning attack under different norm bounds. `C1-FEMNIST`.

impact of the L_2-norm at low norm bounds, which reduces accuracy but much more graceful. This behavior is expected, because the L_∞-norm provides a rigid way to ensure client updates are not too large. With the L_∞-norm, the bound must be the same for both small and large parameters. However, this is problematic because, in machine learning, some updates to parameters are inherently larger than others, depending on the random initialization of the model. The L_2-norm bound provides more flexibility, because it enforces a bound on the total update size. This way, smaller parameters can compensate for larger parameters, which Figure 4.12 shows works better for our networks. Thus, we conclude that the L_2-norm bound is less sensitive to the concrete bound than the L_∞-norm bound. This insight is useful because in many cases, it is impossible to find the optimal norm bound and an approximation must be used instead.

Weight regularization

Existing attacks propose evasion techniques to avoid anomaly detection defenses. One of these techniques is to include a weight regularization term in the objective

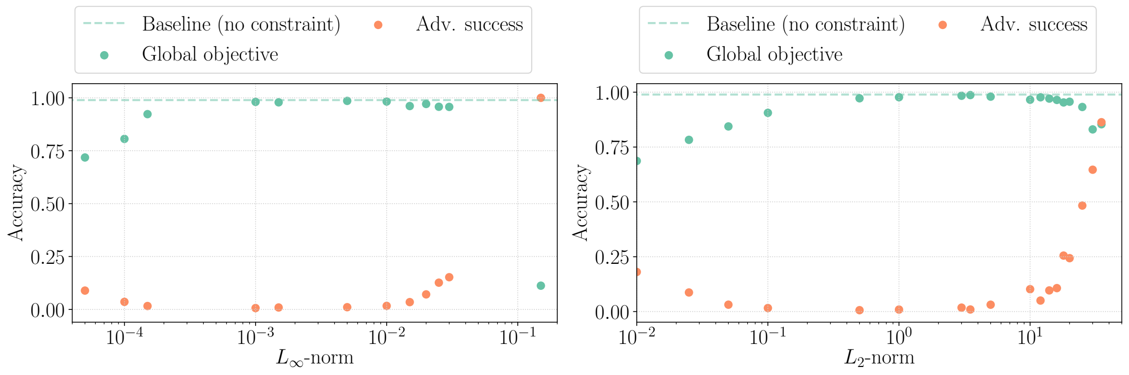


Figure 4.12: Performance of the continuous model poisoning attack under different norm bounds. `C1-FEMNIST`.

that the adversary optimizes. This term ensures that the update stays close to the original model, because otherwise the loss increases. The balance between the malicious objective and this regularization is defined by a weight regularization factor μ with $0 < \mu < 1$. The adversary can vary the value of μ to find a tradeoff between attack effectiveness and detectability. However, in Figure 4.13 we show that under a suitable norm bound, the adversarial success is very low for all values of μ.

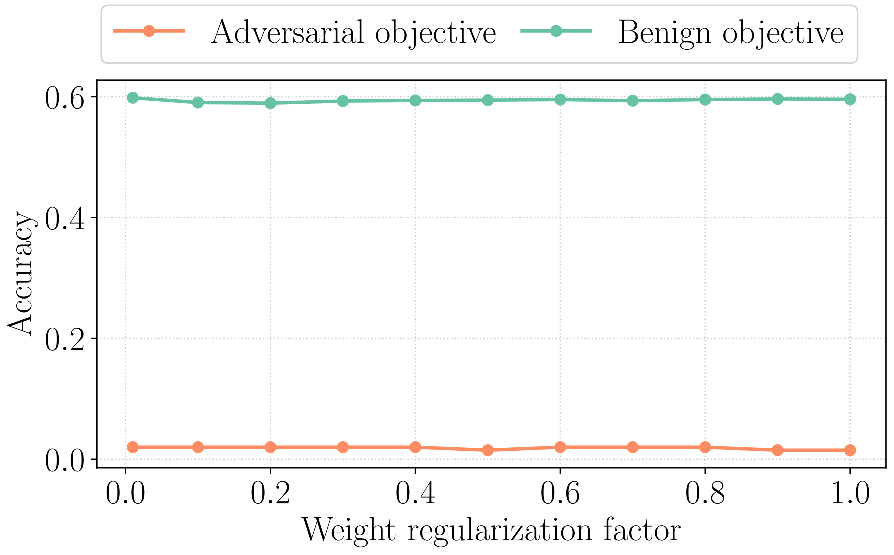


Figure 4.13: Comparison of adversarial success under various weight regularization rates. `C2-CIFAR A3-GREEN`, L_2-norm bound of 0.8.

4.6.3 Improving Attacks

The previous model poisoning attacks did not take the clipping bound into account. We now show how the attacker may attempt to improve this attack, but show empirically that this quickly becomes infeasible in a reasonable federated learning setup.

With a norm bound, the objective for the adversary becomes: Given a clipping bound constraint, find an update that minimizes the cross-entropy loss for the malicious objective that lies within this bound.

$$\underset{\mathbf{x}_m^t}{\text{minimize}} \frac{1}{|\tilde{D}|} \sum_{(\tilde{x},\tilde{y}) \in \tilde{D}} \ell^{xent}(f_{\mathbf{w}_m^t}(\tilde{x}), \tilde{y}, \mathbf{w}_m^t) \text{ s.t. } \mathbf{w}_m^t \in \mathcal{C}$$

Where $\mathcal{C}$ is the set of allowed values for $\mathbf{w}_m^t$ under the norm bound. For instance, if the L_2-norm bound is applied, the attacker must find an update vector $\mathbf{w}_m^t$ that lies within the ball with radius β around $\mathbf{w}_G^t$:

$$\mathcal{C} = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{w}_G^t\|_2 \leq \beta\}$$

However, in federated averaging, the global learning rate θ and the number of clients n directly influence the contribution of a selected client's update:

$$\mathbf{w}_G^{t+1} = \mathbf{w}_G^t + \frac{\theta}{n} \sum_{i \in S} \mathbf{w}_i^{t+1} = \mathbf{w}_G^t + \frac{\theta}{n} \sum_{i \in S \setminus m} \mathbf{w}_i^{t+1} + \frac{\theta}{n} \mathbf{w}_m^{t+1}$$

Because the attacker's update has to survive averaging, we assume the malicious update must take scaling into account. We can therefore rewrite the constraint to the following with scaling factor γ:

$$\mathcal{C} = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{w}_G^t\|_2 \leq \frac{\beta}{\gamma}\}$$

The objective for the attacker is now to find an update that conforms to the norm bound β when scaled by γ

The above minimization problem can be approximated using the Projected Gradient Descent (PGD) algorithm, which is discussed in Section 2.1.4. The PGD algorithm does not provide any guarantees on finding the best possible update. In order to potentially find a better update, PGD may be run multiple runs from the same initial weights. At each run, the initial weights are slightly changed using random noise, to make the algorithm converge to a different local minimum.

Figure 4.14 shows that attacks crafted using PGD are not more effective than regular scaling attacks. The attacks do not influence global model accuracy in comparison to the baseline and are ineffective.

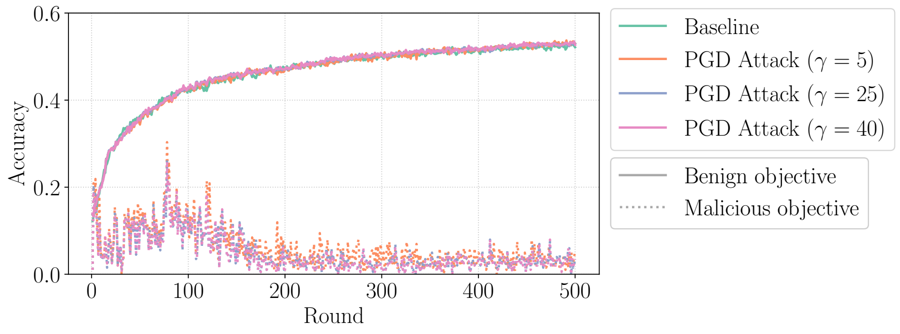


Figure 4.14: Persistent PGD attack every other round under norm bounding defense 0.8. `C2-CIFAR10 A3-GREEN`, LeNet5, 100 clients, 50 selected per round.

4.6.4 Setting the Norm Bound

We have shown that norm bounding that makes it significantly harder to perform model poisoning in a reasonable federated learning setup if an appropriate norm bound is used. We now discuss a method to find an appropriate norm bound for given setup.

The main insight is that the L_2-norm of the benign clients is predictable because the training parameters are the same for all benign clients. Therefore, only the variance in the training data between clients affects the spread of client updates, shown in Figure 4.15.

Given some training data, we can estimate the distribution of client update sizes, by simulating a client performing several rounds of training with the given parameters.

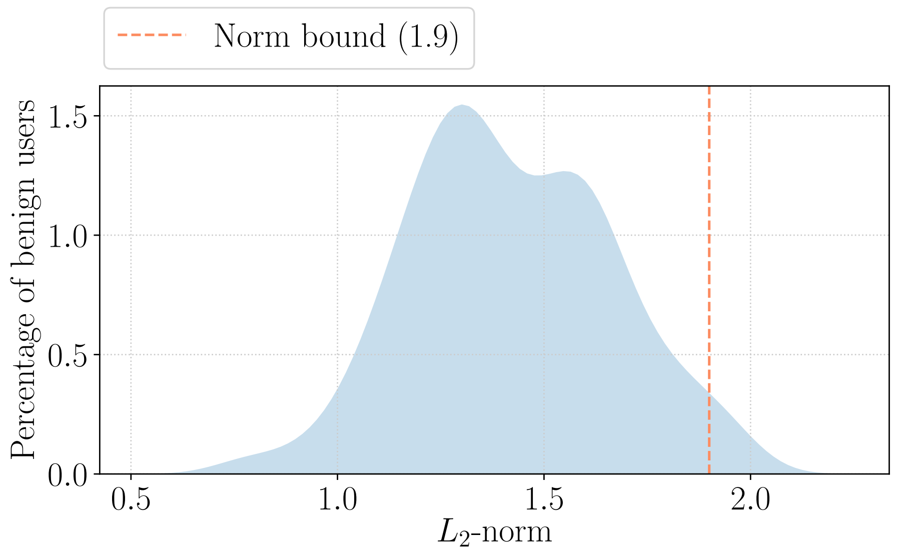


Figure 4.15: Comparison of L_2-norm distribution of benign clients for IID and non-IID data distributions. `C2-CIFAR10`.

We can then set the bound at the $(1 - \epsilon)$-th percentile of the distribution, depending on the expected fraction of adversaries α. If α is high, the norm bound must be tight, so a high ϵ is necessary. This can impact the accuracy of the network slightly by limiting the contributions of the benign clients, so it must be chosen carefully. However, our results from the previous section show that for a reasonable α, the norm bound does not impact model accuracy while successfully preventing attacks. The implementation of the norm bound selection process under secure aggregation is further discussed in Section 5.5.3.

4.7 Conclusion

Model poisoning attacks are a serious threat to the integrity federated learning models. Without any defense, the capabilities for an adversary are disproportionately high in relation to the rest of the clients: An adversary controlling a single malicious client in a network with millions of devices can introduce a backdoor in the model in a single round that can potentially persist for many rounds. We have seen how attacks behave for different training strategies, malicious objectives, scaling factors and regularization techniques. These insights on attacks are important because they allow us to come up with suitable measures that are effective under many conditions. If we do not address these issues, we leave real world federated learning setups, which we increasingly rely upon, vulnerable to integrity attacks.

In this work we show that norm bounding is an effective defense against model poisoning attacks, because malicious updates have to be much larger than benign updates in order to convey the malicious objective in the model and survive the averaging with the updates from other clients. The comparison of the updates of benign clients and attacks show there is a clear difference in update size. Thus, by providing an upper bound on the update size, we can impede the adversary in its attack. Moreover, even when improved attacks such as PGD are used, norm bounding still proves effective.

The use of a norm bound restricts the adversary from only contributing to the model in proportion to the other clients. The attack potential becomes a question of relative contribution: if the attacker can compromise a large enough fraction of clients α, an attack is successful. However, we have empirically shown that even for a very high α of 5%, attacks are impossible under a norm bound. Therefore, we can conclude that norm bounding is an effective defense for model poisoning in a reasonable federated learning setup. This insight can guide us in the development of a protocol for federated learning that is robust against model poisoning attacks.

5 Design

This chapter presents the design of a new protocol for robust privacy-preserving federated learning. The chapter is structured as follows: We start by defining the objectives and threat model we cover in this work. Then, we give a high-level overview of the protocol, after which we review the details for each of our design components. We then discuss how we achieve robustness against malicious client updates. Finally, we show how we optimize the protocol to scale and meet the performance requirements of federated learning workloads.

5.1 Objectives

In Chapter 4 we showed and analyzed how the norm bounding can be an effective solution for model poisoning attacks.

Based on the insight that norm bounding can be effective in limiting adversary success; this holds even in small setups where it is possible to find a norm bound that limits the attacker while not impacting model accuracy. The goal of this thesis is to develop a new secure and robust aggregation protocol for federated learning that effectively incorporate norm bounding for integrity. More specifically, we design a federated learning protocol that fulfills the following requirements:

1. **Privacy-preserving clients' updates:** As individual updates can reveal sensitive information. The individual update $\mathbf{x}_i$ of the clients must be kept private. Formally, when a client update has f-privacy, the aggregation server should be able to learn nothing more about all $\mathbf{x}_i$, except what they can learn from the output of the aggregation function $f(\mathbf{x}_1, ..., \mathbf{x}_k)$.

2. **Robustness:** The aggregation protocol should resilient to poisoning attacks from malicious clients.

3. **Scalability:** The protocol must be efficient enough to deploy at scale, in particular for a large number of clients and for models with a reasonable number of parameters.

Similar to the setup used in the analysis in Chapter 4, we assume a typical federated learning setup containing a server, multiple clients, and an adversary that can compromise one or more clients. The parties iteratively train a shared global machine learning model on a shared task of which the clients individually hold a potentially disjoint, non-IID subset of the training data. The iterative process is divided in rounds, in which the server shares the current global model with the clients, the clients train the model locally and then submit an update to the server. At the end of the round, the server aggregates all the client updates and updates the shared global model.

5.1.1 Threat Model

The protocol is designed under the assumption that both clients and the server can misbehave. We assume the server to be *honest-but-curious*, i.e., the server follows the protocol correctly but will analyze all observed data to gain as much information as possible. This is not such a far-fetched belief because in federated learning the server is often controlled by the service provider who has an interest in providing a well-functioning service. The server follows the protocol correctly but may try to gain additional profit by trying to collect as much data as possible from the clients. We assume that the server can store and analyze all messages it gets from clients. Additionally, the server is not able to inspect the client's local training data or influence the training process. We assume that the clients and server communicate over a secure channel and clients and the server are authenticated using a Public Key Infrastructure (PKI).

On the client-side, we assume at most a fraction α of clients can be compromised by the adversary. While benign clients do not know of each others existence, clients compromised by the adversary can communicate and exchange information freely. Malicious clients can deviate from protocol at any point by withholding, malforming, or replaying messages. However, due to the existence of the PKI, the attacker can not impersonate or simulate arbitrary parties. Instead, we focus particularly on the attack where the goal of the adversary is to make the global model converge to a poisoned version in which it performs some malicious task. To this end, malicious clients may propose compromised model updates that will infect the global model. Because the setup uses a secure aggregation protocol, the contribution of each client into the global aggregated model cannot be attributed to an individual client, severely impairing the server's defensive possibilities. Similar to the *honest-but-curious* server, the adversary can not access the training data of the benign clients, nor inspect the training process. The adversary does have full knowledge of the configuration of the federated learning system such as the training hyperparameters and number of clients, following Kerckhoff's principle.

Out of scope attacks.

This work does not consider the following attacks that are possible under our threat model:

- Attacks on privacy through inspecting the parameters of the global model $f(\mathbf{x}_1, ..., \mathbf{x}_k)$. While our protocol hides individual client updates, the global model that is shared with all clients and the server may still leak information of the clients' dataset, even after aggregation. In order to overcome this, techniques such as differential privacy [39] may be used. In differential privacy, the disclosure of an individual client's private information in some aggregate statistics or machine learning model is theoretically limited. This is done by adding sufficient random noise to the model. For a formal definition of differential privacy, we refer to Appendix A. Much work on this topic already exists [20,50,95,103] and can be applied complementary to our protocol. As this work focuses on model integrity in federated learning under secure aggregation, we deem the privacy of the shared model out of scope.

- Attacks through side-channels such as timing attacks. Timing attacks measure the time it takes for a client to perform cryptographic operations in an attempt to infer sensitive information. In other fields, it has been shown [67] that timing attacks can be very subtle by statistically filtering out the influence of other operations based on time, making it possible to precisely measure the duration of the operations that reveal the sensitive information. Timing attacks can be prevented using algorithms that rely on constant-time cryptographic operations. Side-channel attacks are a separate field of study and ways to mitigate such attacks can in principle be applied to our protocol.

- Denial-of-Service attacks. Clients may behave in such a way that prevents the protocol from giving an output, by providing malicious blinding values or withholding messages. This malicious behavior is detectable in the protocol by the server, so it can never cause a wrong protocol output. Upon detection, the server can abort the round and start over. In practice, the impact of the Denial-of-Service attack vector is limited, as it can be easily mitigated, i.e., upon the detecting of such an attack, the server can restart the round with a different set of clients. Furthermore, because the clients are all uniquely identifiable, the server can employ a system to keep track of which clients were included in which rounds in an attempt to identify the compromised clients and exclude them from the system.

5.2 Protocol Overview

Our approach makes use of a homomorphic commitment scheme together with cryptographic proofs to achieve privacy-preserving aggregation and guarantees for model integrity. In the previous chapter, we show that norm bounding can be an effective mechanism to ensure integrity without compromising accuracy. In our protocol, we resort to the use of cryptographic proofs to enforce and ensure that the received encrypted updates conform to our desired norm bound.

Client updates are encrypted using the homomorphic commitment scheme and aggregated at the server. In addition, clients submit cryptographic proofs for the commitments attesting that their update lies within the given norm bound. The server then decrypts the aggregate and verifies the cryptographic proofs the ensure the updates lie within the norm bound range.

We now give a high-level overview of the protocol, and then go on to explain core components such as update aggregation and verification in detail. The protocol consists of a *setup* phase and a *training* phase.

Setup Phase. In the setup phase, the server shares the configuration of the federated learning setup $\mathbb{C}$ with the clients. This configuration contains the properties for the model, training parameters, and the security parameters. After receiving the required configuration, the clients enter into the training phase.

Training Phase. The training phase is organized in rounds. In each round, the server and clients communicate in a sequence of steps:

1. **Weight download:** The server selects a subset of clients K to participate in the current training round. The server then transmits the model weights $\mathbf{w}_G^t$ to all the participating clients K, along with the required norm bound B.
2. **Client training:** The clients compute a local update $\Delta\mathbf{w}_i^{t+1}$ using the local training algorithm. They ensure the update conforms to the bound B by clipping or scaling if the update exceeds the bound.
3. **Update aggregation:** The clients encrypt their update and perform a secure aggregation protocol together with the server, resulting in an aggregated update $\Delta\mathbf{w}_G^{t+1}$ as output at the server.
4. **Update verification:** Every client proofs that their update lies within the given norm bound. To this end, each client generates a set of zero-knowledge proofs $\mathbf{R}_i^{t+1}$ that proofs this norm bound and sends these to the server. The server verifies correctness of the zero-knowledge proofs for each client.
5. **Weight update:** The server applies the update to the global model together with the global learning rate θ: $\mathbf{w}_G^{t+1} \leftarrow \mathbf{w}_G^t + \theta\Delta\mathbf{w}_G^{t+1}$

The protocol is visualized as a sequence diagram in Figure 5.1. The protocol terminates after T rounds, or after a specific accuracy has been reached by the global model.

5.3 Update Aggregation

In essence, the aggregation protocol in our setup is similar to the general federated learning aggregation protocol but includes additional steps that ensure confidentiality and integrity. After performing local training, each client holds a local model update $\Delta\mathbf{w}_i$ that it wants to send to the server. To ensure client update f-privacy, clients encrypt their updates in a way that the server can only have access to the output

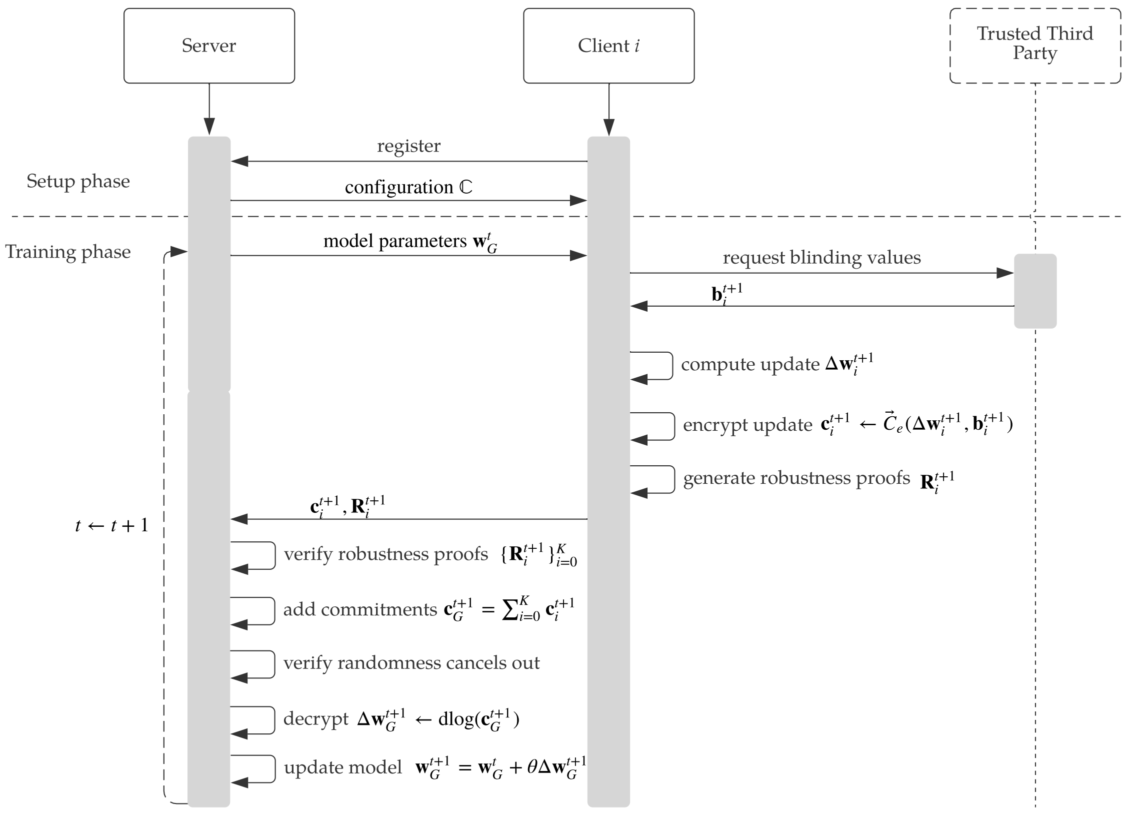

Server
Client i
Trusted Third Party
register
Setup phase
configuration $\mathbb{C}$
Training phase
model parameters $\mathbf{w}_G^t$
request blinding values
$\mathbf{b}_i^{t+1}$
compute update $\Delta\mathbf{w}_i^{t+1}$
encrypt update $\mathbf{c}_i^{t+1} \leftarrow \vec{C}_e(\Delta\mathbf{w}_i^{t+1}, \mathbf{b}_i^{t+1})$
generate robustness proofs $\mathbf{R}_i^{t+1}$
$\mathbf{c}_i^{t+1}, \mathbf{R}_i^{t+1}$
$t \leftarrow t+1$
verify robustness proofs $\{\mathbf{R}_i^{t+1}\}_{i=0}^K$
add commitments $\mathbf{c}_G^{t+1} = \sum_{i=0}^K \mathbf{c}_i^{t+1}$
verify randomness cancels out
decrypt $\Delta\mathbf{w}_G^{t+1} \leftarrow \mathrm{dlog}(\mathbf{c}_G^{t+1})$
update model $\mathbf{w}_G^{t+1} = \mathbf{w}_G^t + \theta\Delta\mathbf{w}_G^{t+1}$

Figure 5.1: Sequence diagram of the protocol.

aggregate of all the client updates $\Delta\mathbf{w}_G$ but cant see any individual update. To ensure confidentiality and integrity, we design a secure aggregation protocol that is specifically tailored to allow the enforcement of norm bounding.

First, we explain how the privacy of client updates is preserved with regards to the honest-but-curious server using a secure aggregation protocol. Then, we present how the vanilla secure aggregation protocol can be made resistant to abuse by actively adversarial clients.

5.3.1 Secure Aggregation

Notation: In the rest of the chapter, we denote variables that are vectors with **bold** symbols $\mathbf{w}$ and the ath element with a superscript w^a. A variable that belongs to client i is denoted with a subscript $\mathbf{w}_i$, and variables that are global with a G subscript $\mathbf{w}_G$. Additionally, the elementwise exponentiation of a vector $\mathbf{w}$ with g is denoted as $g^{\mathbf{w}}$. Finally, $[x]^D$ denotes a vector of D elements x.

Protocol: Our protocol operates on a group $\mathbb{G}$ of prime order p with publicly known generators $g, h \in \mathbb{G}$. We assume the parameters in the clients' model update are elements of $\mathbb{Z}_p$, i.e. $\Delta\mathbf{w}_i \in \mathbb{Z}_p^D$. At the start of secure aggregation, every client generates a commitment to their update vector $\Delta\mathbf{w}_i$ using the ElGamal commitment function. A more elaborate description of ElGamal commitments can be found in Section 2.2.2. In our protocol, the purpose of the ElGamal commitments is two-fold. First, they are used to hide the update $\Delta\mathbf{w}_i$ from the server. Second, the commitments allow the clients to prove the norm bound in zero-knowledge later on in the protocol, because of the binding property. Given a vector of blinding values $\mathbf{b}_i \in \mathbb{Z}_p^D$ of the same size as $\Delta\mathbf{w}_i$, a client commits to their update $\Delta\mathbf{w}_i$ as follows:

$$\vec{C}_e(\Delta\mathbf{w}_i, \mathbf{b}_i) = (g^{\Delta\mathbf{w}_i} h^{\mathbf{b}_i}, g^{\mathbf{b}_i})$$

where $\vec{C}_e$ denotes the elementwise application of the ElGamal commitment function C_e. For now, we assume every client i receives a vector of random blinding values $\mathbf{b}_i$ from a trusted third party that cancel out when added together:

$$\sum_{i \in K} \mathbf{b}_i = \mathbf{0}^D$$

where $\mathbf{0}^D$ denotes a vector of length D consisting of all zeroes. We describe how clients attain this vector of canceling blinding values without the requirement of a trusted third party in Section 5.3.3.

Every client encrypts their update vector $\mathbf{c}_i \leftarrow \vec{C}_e(\Delta\mathbf{w}_i, \mathbf{b}_i)$ and sends $\mathbf{c}_i$ to the server. The server then adds all the vectors of the homomorphic ElGamal commitments $\mathbf{c}_i$ element-wise together:

$$\begin{aligned}\prod_{i\in K}\mathbf{c}_i &= \vec{C}_e(\sum_{i\in K}\Delta\mathbf{w}_i, \sum_{i\in K}\mathbf{b}_i)\\ &= \vec{C}_e(\sum_{i\in K}\Delta\mathbf{w}_i, \mathbf{0}^D)\\ &= \left[(g^{\sum_{i\in K}\Delta w_i}h^0, g^0)\right]^D\\ &= \left[(g^{\sum_{i\in K}\Delta w_i}, g^0)\right]^D\end{aligned}$$

The server is then left with a vector with tuples of two group elements for each parameter in the model. The first element of tuple i contains parameter w^i of the global model update, i.e. the sum of this parameter of all the client updates $\Delta w_G^i = \sum_{k\in K}\Delta w_k^i$, and the second element contains the neutral element.

$$\left[(g^{\sum_{i\in K}\Delta w_i}, g^0)\right]^D = \left[(g^{\Delta w_G}, g^0)\right]^D = \begin{bmatrix}(g^{\Delta w_G^1}, g^0)\\ (g^{\Delta w_G^2}, g^0)\\ \vdots\\ (g^{\Delta w_G^{D-1}}, g^0)\\ (g^{\Delta w_G^D}, g^0)\end{bmatrix}$$

Decrypting the update. The last step of the secure aggregation is the decryption of the aggregated ElGamal commitments vector $(g^{\Delta w_G}, g^0)^D$ by the server to retrieve the global model update $\Delta\mathbf{w}_G$. In order to do this, the server has to compute, for every tuple, the discrete log of $g^{\Delta\mathbf{w}_G}$. Recall that we assume that the discrete log is hard for our group $\mathbb{G}$. However, we have an expectation of the value of $\Delta\mathbf{w}_G$, making the computation of the discrete log using a non-polynomial algorithm feasible, as we only have to check the values that are close to our expectation. Concretely, in our case the expectation is that $\Delta\mathbf{w}_G$ is at most 8-, 16- or 32-bits long, which is feasible to compute using an algorithm such as baby-step giant-step [98], which has a running time of $O(\sqrt{n})$.

5.3.2 Active Adversarial Clients

We have described how our protocol guarantees f-privacy of the client's updates and works correctly when clients are honest. We now explain how our protocol protects against dishonest clients

Recall from the security definition of our secure multi-party protocol that a protocol is secure if the adversary cannot achieve anything in the protocol that he could not achieve in the specification [79]. Therefore, in the next paragraphs, we focus on how the protocol prevents actively malicious clients from deviating from the secure aggregation protocol, but not how it prevents malicious clients from submitting a malicious update $\Delta\hat{\mathbf{w}}_i$ to perform a model poisoning attack. Preventing model poisoning attacks using norm bounding is discussed in Section 5.4.

Invalid blinding values. Because actively malicious clients may deviate from the protocol at any time, an adversary can submit malicious blinding values that do not cancel out with the other blinding values. If the blinding values do not cancel out, the commitment containing aggregate model update $\Delta\mathbf{w}_G$ is compromised and will provide no useful information:

$$g^{\sum_{i\in K}\Delta\mathbf{w}_i} h^{\sum_{i\in K}\mathbf{b}_i} \neq g^{\sum_{i\in K}\Delta w_i} h^0$$

Furthermore, it will be very likely that decryption is impossible because the update now lies outside the expected range of values. To protect against this attack, the server can verify that the second element in the ElGamal commitment tuple is equal to the neutral element g_0 for all parameters. If this is not the case, the server knows the blinding values do not cancel out and can restart the round with a different subset of clients.

While the server can not prevent malicious clients from submitting invalid blinding values, the protocol does prevent the output of garbage values by making abuse detectable. Furthermore, in the case that repeated attacks occur, the detectability allows the server to get an intuition on which client is malicious. For even more resiliency against this type of attack, the server can keep track of which clients participated in which failed rounds. This way, the server is able to discover which client is responsible for the abuse and exclude this client from the protocol. For this reason, we deem this attack vector on the protocol acceptable, because the impact is limited as the attack is detectable and can be mitigated accordingly.

Correctness of the ElGamal commitments. We have seen that submitting the wrong blinding values in the ElGamal commitments is detectable by the server. However, an adversary is also able to submit a completely invalid ElGamal commitment. Specifically, an adversary could submit a commitment,

$$(g^w h^{b'}, g^b)$$

where b is the correctly canceling blinding value, and b' is a malicious blinding value such that $b' \neq b$. Now, the check at the server of the second parameter will succeed, while still compromising the global model update. To overcome this, we need a guarantee that the ElGamal commitment was constructed correctly, i.e. that the blinding values used in both elements of the commitment are the same, $b' = b$.

To this end, we use a zero-knowledge randomness proof, or randomness proof. With the randomness proof, given an ElGamal commitment $(c_l, c_r) \in \mathbb{G}^2$, the client proofs that they know an opening (x, r) of the commitment such that $(c_l, c_r) = (g^x h^r, g^r)$. Formally, the property is captured in the predicate:

$$Q((x,r),(g,h,(c_l,c_r))) = ((c_l,c_r) \stackrel{?}{=} (g^x h^r, g^r)$$

This is implemented using a Sigma protocol. In Section 5.4, we show the randomness proof is integrated in the update verification step of the protocol.

Client Dropout In a setup with a set of loosely federated, unreliable clients in an unreliable network, client dropout can be a regular occurrence. Our protocol can handle client dropouts following the solution proposed by Bonawitz et al. [23]. In this approach, clients share their blinding values using a threshold secret sharing scheme with other clients. When a client does not submit their update to the server, the other clients can reconstruct the blinding value of the dropped client, as long as there are more benign clients than the threshold.

However, only secret-sharing the pairwise blinding values introduces another security problem: The server can lie and claim a client has not sent their update, triggering the reconstruction of the users blinding value. With this blinding value and the client's commitment, the server can now access the secret value of the client. To prevent abuse from the honest-but-curious server, clients use a secondary, private mask, which is also secret shared. Benign clients only reveal a share for one of the two blinding values per client, and not both. This forces the server to make a choice in the aggregation round, namely to ask for a share of the pairwise shared mask, or for a share of the private mask for every client.

5.3.3 Removing the trusted setup

To improve readability, we assumed the existence of a trusted third party to distribute the vectors of element-wise canceling blinding values $\mathbf{b}_i$. We now explain how to remove the need for a trusted third party by performing some additional steps in the setup phase.

At a high level, every pair of clients (u, v) in the set of clients K for $u < v$ agrees through some protocol on a common random vector $\mathbf{s}_{u,v}$. Note that we assume some kind of topological ordering on the set of clients K. This is in practice achievable in many ways, for instance, by using the client's public key as input. With these shared vectors, each client computes their own vector of blinding values $\mathbf{b}_i$. We first explain how a client u creates their blinding vector $\mathbf{b}_i$, given the random vectors $\mathbf{s}_{u,v}$ shared with every other client. Afterwards, we describe how two clients agree on these random vectors.

To create blinding values that cancel out when added together, clients compute the following. Recall that a client u already shares a random vector with every other client, which can be expressed as $\{\mathbf{s}_{u,v} \mid u, v \in K \land u < v\} \cup \{\mathbf{s}_{v,u} \mid u, v \in K \land u > v\}$. A client u then computes their vector of blinding values by adding the random vectors shared with clients of a lower order and subtracting the random vectors shared with clients of a higher order.

$$\mathbf{b}_u = \sum_{v \in K : u < v} \mathbf{s}_{u,v} - \sum_{v \in K : u > v} \mathbf{s}_{u,v}$$

The blinding values $\mathbf{b_u}$ cancel out when they are added together. To prove this, we show that each shared random vector is added and subtracted exactly once:

$$\begin{aligned}\sum_{u \in K} \mathbf{b}_u &= \sum_{u \in K} \left(\sum_{v \in K: u<v} \mathbf{s}_{u,v} - \sum_{v \in K: u>v} \mathbf{s}_{u,v} \right) \\ &= \sum_{(u,v) \in (K \times K): u<v\}} (\mathbf{s}_{u,v} - \mathbf{s}_{u,v}) \\ &= \vec{\mathbf{0}}\end{aligned}$$

Shared random vector agreement. To establish a shared random vector, clients engage in a Diffie-Hellman key agreement protocol as described in Section 2.2.3. If this is done naïvely, this protocol step incurs an overhead with a complexity of $O(K^2D)$ per round where D is the number of parameters in the model and K is the number of clients. However, two optimizations can be applied to reduce the complexity to $O(K^2)$ and to allow the agreement protocol to run only once.

First, instead of performing Diffie-Hellman key agreement for each parameter, two clients can agree on a single value that is then used as an input seed to a Pseudo-Random Generator (PRG). The PRG then expands the randomness of the single value to a full vector of values. This reduces the amount of key agreement protocol executions from D times to just once.

Second, we can use the shared random values of round t in combination with a cryptographic hash function to generate the shared random values of round t. Because of the deterministic property of the hash function, we only need to perform key agreement once in the first round.

In conclusion, we can use Diffie-Hellman key agreement in the setup phase for clients to agree on pairwise shared vectors which are then converted into canceling blinding values used in aggregation. This removes the need for a trusted third party at an additional cost of $O(K^2)$ at the start of the protocol.

5.4 Update Verification

The goal of update verification is to constrain the impact of malicious contributions on the aggregation process. A client compromised by the adversary can send a malicious update $\Delta\hat{\mathbf{w}}$ to the server to perform a model poisoning attack. In Chapter 4, we showed that norm bounding is an effective defense against model poisoning. Norm bounding prevents the adversary from performing model poisoning attacks by limiting the size of the contribution to the model per client. In this section, we explain how we implement norm bounding of the client updates $\|\mathbf{x}\|_p \leq B$ using zero-knowledge proofs, with B defined by the server .

Our protocol supports two kinds of norm bound p: The L_2- and the L_∞-norm. We start by explaining how the L_∞-norm bound is constructed, which only requires a single type of zero-knowledge proof: parameter-wise range proofs. Afterwards we review the construction of the L_2-norm bound, which needs two types of zero-knowledge proofs in addition to the parameter-wise range proofs: proof of square commitments and a single L_2-norm range proof.

5.4.1 L_∞-Norm

For the L_∞-norm bound, each parameter of the client's update vector $\Delta\mathbf{w}$ is constrained to a given bound. To achieve the L_∞-norm bound, we only need to prove that every parameter lies within the bound using a parameter-wise range proof. Given the update $\Delta\mathbf{w}_i$ from client i containing the vector of ElGamal commitments $[(g^x h^r, g^r)]^D$, Formally, given a bound B, we require every parameter $x \in \Delta\mathbf{w}$ to be within the range $[-B, B]$. We use the Bulletproof [26] range proofs that provide fast proof generation and verification. Furthermore, the proof size is small and logarithmic in the number of bits n in the range. Bulletproofs also support batching of proofs for the same range, with the proof size growing logarithmically in the number of proofs.

The cryptographic proof validates that the value lies within a range that is a power of 2, i.e. $[0, 2^n]$. To be able to proof a parameter lies in the $[-B, B]$ range, we first choose an n such that $2^{n-1} < B \leq 2^n$. Then, the clients homomorphically shift the commitments to the parameters $g^x h^r$ by $g^{2^{n-1}}$ and proof that the shifted parameters $x + 2^{n-1}$ lie in the range $[0, 2^n]$.

5.4.2 L_2-Norm

As shown in Section 4.6.1, the L_2-norm bound is less sensitive to the concrete norm bound value than the L_∞-norm. This is because the L_∞-norm provides a more rigid way to ensure client updates are not too large by enforcing the same limit for every parameter, whereas the L_2-norm enforces a bound on the total update size.

The previously described L_∞-norm for D parameters implicitly bounds the L_2-norm to $\sqrt{K2^{(n-1)^2}}$ because the parameter vector can at most have an L_2-norm of this size. However, the bound is not a t as the true L_2-norm, because the lower parameters in the vector can not compensate for the higher parameters. In addition to this flexibility, the construction of the L_2-norm has two practical advantages over the L_∞-norm.

First, it is better compatible with some scalability improvements. Some of these improvements, such as random subspace learning, rely on a random transformation of the weight matrices. The transformation matrix has orthogonal columns, which means that it preserves the inner product between two vectors, which implies the L_2-norm of the vector is preserved. We further elaborate on this in Section 5.6.

Second, while the overhead for both the L_2-norm and the L_∞-norm are similar, the L_2 allows for better fine-tuning of the norm without much additional overhead. This is because it takes only a single range proof for the actual L_2-norm bound itself. This point will become more clear after reviewing how the L_2-norm is constructed, we discuss this further in Section 5.5.2.

We now explain how the L_2-norm bound proof is constructed. Instead of bounding the L_2-norm by a bound B, which contains a square root, we bound the squared L_2-norm by B^2.

$$\|\mathbf{x}\|_2 \leq B \iff \|\mathbf{x}\|_2^2 \leq B^2$$

This is possible because both the norm and the bound are always positive and it lets us avoid an expensive secure square-root operation. The squared L_2-norm of a vector $\mathbf{x} \in \mathbb{R}^D$ is defined as:

$$\|\mathbf{x}\|_2^2 = \left(\sqrt{\sum_{i=1}^{D} x_i^2}\right)^2 = x_1^2 + \cdots + x_D^2$$

The proof for the L_2-norm consists of three components.

1. **Square commitments.** For each parameter x, a commitment to the square of x, $g^{x^2}h^r$, as well as a zero-knowledge proof that states that the commitment to x and the commitment to x^2 are in fact the same x.
2. **L_2-norm range proof.** A range proof to prove that the sum of the commitments to the squares is within $[0, B^2]$, which is exactly the squared L_2-norm.

(3.) **Parameter-wise range proof.** The parameter-wise range proof for each parameter x, which we also used for the L_∞-norm, declaring that it is within $[-B, B]$. These parameter-wise range proofs are required to ensure that no arithmetic overflows occur, i.e., if a client submits a large x that overflows the L_2-norm in the 256-bit group bringing below the bound B. However, because they exist to only prevent overflow and not the exact norm bound itself, the actual bound B does not have to be very precise, because the L_2-norm range proof already bounds the update norm.

We now review the two additional components required for the L_2-norm proof. For simplicity in the next section, we provide an explanation for a single parameter x, but the operations are done to every parameter in the update vector.

1. Square commitments: Recall from the definition of the L_2-norm that in order to compute L_2-norm, we need to get the square of each parameter x. Because the partially homomorphic ElGamal commitments only support homomorphic addition, it is impossible to calculate the square of x using our existing commitment to x, $(g^x h^{r_1}, g^{r_1})$. Therefore, the client has to perform this calculation and send an additional commitment to the square of x. The x^2 is in this case is sent as a Pedersen commitment, as we do not need the second group element that represents the randomness. In fact, the client can use any randomness r_2 in the Pedersen commitment to x^2.

Along with this extra commitment, the client must submit a zero-knowledge proof that proves that the computation was performed correctly. In other words, the client proves that the same x is used to construct the ElGamal commitment $(g^x h^{r_1}, g^{r_1})$ as well as the Pedersen commitment $g^{x^2} h^{r_2}$. For efficiency reasons, we incorporate this zero-knowledge proof of knowledge in the already existing randomness proof, referring to it as the squared randomness proof. We refer to this as the squared randomness proof and the exact details are described in Section 5.4.3.

2. L_2-norm range proof: With the commitments to the square of the parameters $g^{x_i^2} h^{r_{2,i}}$, the server can compute the squared L_2-norm using the homomorphic property of the Pedersen commitments:

$$\begin{aligned}\prod_{i=1}^{D} g^{x_i^2} h^{r_{2,i}} &= g^{\sum_{i=1}^{D} x_i^2} h^{\sum_{i=1}^{D} r_{2,i}} \\ &= g^{\|\mathbf{x}\|_2^2} h^{\sum_{i=1}^{D} r_{2,i}} \\ &= c_{L_2}\end{aligned}$$

What remains is for the client to provide a zero-knowledge proof that $\|\mathbf{x}\|_2^2 \leq B^2 = 2^{n^2}$ using a range proof for the predicate:

$$Q((\|\mathbf{x}\|_2^2, \sum_{i=1}^{D} r_{2i}), (g, h, c_{L_2})) = \left(c_{L_2} \overset{?}{=} g^{\|\mathbf{x}\|_2^2} h^{\sum_{i=1}^{D} r_{2i}} \wedge \|\mathbf{x}\|_2^2 \in \left[0, 2^{n^2}\right]\right)$$

We implement this single range proof using a Bulletproof range proof. The server verifies this zero-knowledge proof together with its own computation of c_{L_2} to verify that the L_2-norm proof is correct. We now have a full construction of the L_2-norm bound zero-knowledge proof.

5.4.3 Squared Randomness Proof

At this point, there are two things left to prove. First, as explained in Section 5.3.2, we need to prove that the parameter ElGamal commitments $(g^x h^r, g^r)$ contain the same randomness r within the commitment tuple, which we refer to as the randomness proof. Second, for the L_2-norm, the client needs to provide a proof of square relation, i.e., show that the commitment $g^{x^2} h^r$ contains the same x as in the parameter commitment $g^x h^r$. Because both proofs operate on the same ElGamal commitment, and are proofs of knowledge, we can efficiently combine them into a single Sigma protocol. Note that the second proof is only required for the L_2-norm proof and for the L_∞-norm, only the randomness proof is needed. We can therefore use a subset of the squared randomness proof in the case of the L_∞-norm. We now explain how the squared randomness proof works.

Proof of Square Relation

We know that we can create a Sigma protocol to proof knowledge of the pre-image of any group homomorphism f [80]. However, considering our function to commit to the square of x,

$$f : \mathbb{Z}_p^2 \to H : (x, r) \mapsto g^{x^2} h^r$$

is not homomorphic, because:

$$\begin{gathered} f(a+b, r+r) = f(a,r)f(b,r) \iff \\ g^{(a+b)^2} h^{2r} = g^{a^2+b^2} h^{2r} \iff \\ (a+b)^2 \neq a^2 + b^2 \end{gathered}$$

To solve this, we can proof a slightly different relation: Given $(E_l, E_r) = (g^x h^{r_1}, g^{r_1})$ and $P = g^{x^2} h^{r_2}$, we can rewrite P in the basis of E_l instead of g

$$P = g^{x^2} h^{r_2} = g^{x^2} h^{xr_1} h^{r_2} h^{-xr_1} = (g^x h^{r_1})^x h^{r_2} h^{-xr_1} = {E_l}^x h^{r_2 - xr_1}$$

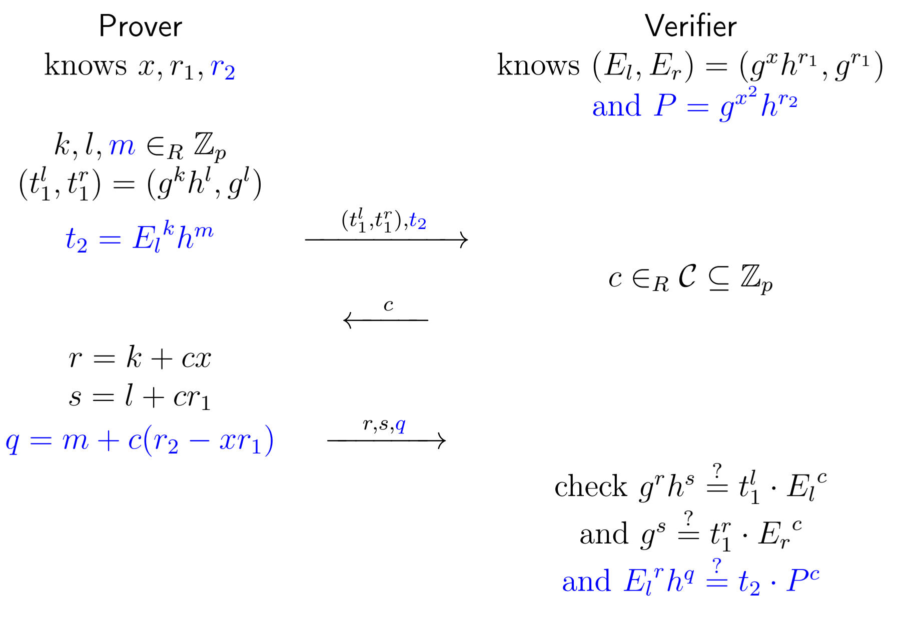

Prover
knows x, r_1, r_2
$k, l, m \in_R \mathbb{Z}_p$
$(t_1^l, t_1^r) = (g^k h^l, g^l)$
$t_2 = {E_l}^k h^m$
$\xrightarrow{(t_1^l, t_1^r), t_2}$
Verifier
knows $(E_l, E_r) = (g^x h^{r_1}, g^{r_1})$
and $P = g^{x^2} h^{r_2}$
$c \in_R \mathcal{C} \subseteq \mathbb{Z}_p$
$\xleftarrow{c}$
$r = k + cx$
$s = l + cr_1$
$q = m + c(r_2 - xr_1)$
$\xrightarrow{r,s,q}$
check $g^r h^s \overset{?}{=} t_1^l \cdot {E_l}^c$
and $g^s \overset{?}{=} t_1^r \cdot {E_r}^c$
and ${E_l}^r h^q \overset{?}{=} t_2 \cdot P^c$

Figure 5.2: Squared randomness proof protocol. The parts marked in black correspond to the randomness proof, whereas the parts marked in blue are only required for the proof of the square relation for the L_2-norm.

Protocol

The insight on the square relation can be incorporated in the already existing randomness proof protocol. The full squared randomness proof is shown in Figure 5.2, and works as follows:

1. The prover picks the values k, l, m at random from $\mathbb{Z}_p$ and computes the ElGamal commitment to (k, l) and the Pedersen commitment of (k, m) in base E_l.
2. The verifier picks a challenge c at random from the challenge space $\mathcal{C} \subseteq \mathbb{Z}_p$ and sends it to the prover
3. The prover computes the response r, s, q based on the challenge and the values it has committed to in the first round, k, l, m. The prover then sends r, s, q to the verifier.
4. The verifier checks the equality of the three equations. With this step, the verifier checks whether the randomness in the two elements of the ElGamal commitment match, and whether the value x used in the ElGamal and in the

Security Proof

We now proof the correctness and security of the squared randomness proof. As established by Maurer [80], a sound, proof of knowledge honest-verifier zero-knowledge protocol can be established for any one-way group homomorphism, provided that two additional properties are met as stated in Theorem 2.2.3. If we can show that our protocol can be seen as a proof of knowledge of the pre-image of a one-way group homomorphism f and proof two additional properties of Theorem 2.2.3, we get the security properties for free.

In our case, we consider the following two functions. f_1 is the ElGamal commitment function and f_2 is a commitment to the square of x:

$$f_1 : \mathbb{Z}_p^2 \to H^2 : (x, r) \mapsto (g^x h^r, g^r) \tag{5.1}$$

$$f_2 : \mathbb{Z}_p^2 \to H : (x, r) \mapsto g^{x^2} h^r \tag{5.2}$$

Clearly, f_2 is not a group homomorphism. However, we can rewrite f_2 in the base of the first output of f_1, as such:

$$f_2' : \mathbb{Z}_n^2 \to H : (x, r) \mapsto (g^{x'} h^{r'})^x h^{r - xr'})$$

f_2 is now homomorphic. We can combine two separate homomorphic functions into one[1]:

[1]Note that in the case of the L_∞-norm bound proof, only the randomness proof part is required, which is means f_2 is not used. It then suffices to set $F = f_1$, and the rest of the formal proof still holds if we set $u = (0, 0)$.

$$F : \mathbb{Z}_n^3 \to H : (x, r_1, r_2) \mapsto (g^x h^{r_1}, g^{r_1}, (g^{x'} h^{r'})^x h^{r_2 - x r'})$$

The function F is a one-way homomorphic function. Additionally, if we set $\ell = p$ and $u = (0, 0, 0)$, Theorem 2.2.3 applies. From this it follows that F is *2-extractable* and *c-simulatable*. Therefore, our protocol is a proof of knowledge and *perfect* zero-knowledge.

5.5 Practical Considerations

In this section we discuss three practical considerations for the protocol. First, we explain how we convert our parameters between machine learning and cryptographic formats. Second, we discuss how we relax the constraint of the norm bound having to be a power of 2 to arbitrary bounds. Finally, we describe a method to select a suitable norm bound in our protocol.

5.5.1 Fixed-point Representation

The machine learning architecture makes use of numbers in floating-point format. However, the cryptographic primitives that we rely on for aggregation require elements from $\mathbb{Z}_p$ as input. Hence, there is a need to convert from floating-point to integer representation and back. In order to achieve this, we use a fixed-point representation of the floating-point numbers.

Fixed-point representations encode numbers with the decimal point at a fixed position, the phenomenon from which it lends its name. We refer to a fixed-point representation of n bits using m decimal bits as a (n, m)-fixed-point encoding. An example of a number encoded in fixed-point representation is shown in Figure 5.3. Starting from a floating-point number, the number is rounded to the closest number in the fixed representation. Furthermore, if the number lies outside of the range of fixed-point representation, it is clipped to the closest extreme. Information may be lost in this quantization process, which we quantify in Section 7.4.1. A schematic overview of the conversion process is as follows:

$$\text{float} \xrightarrow{\text{quantize}} \text{fixed-point} \xrightarrow{\text{interpreted as}} \mathbb{Z}_p$$

The use of fixed-point representation impacts the meaning of the range proofs, because the first m bits encode decimal numbers. Therefore, instead of proving that a value lies in the range $[0, 2^n]$ for all $n \in \mathbb{N}$, it actually shifts the range to $[0, 2^{n-m}]$. For

$$\underbrace{1010}_{12} \;\textbf{.}\; \underbrace{0010}_{+\ 0.125}$$

Figure 5.3: The number 10.125 in (8,4)-fixed-point representation

example in our (8,4)-fixed-point encoding, if we want to prove a range of $n = 2$ bits, we are actually proving that the value lies in the range $[0, 2^{2-4}] = [0, \frac{1}{2^2}] = [0, 0.25]$.

Arithmetic. The numbers, after being converted to fixed-point representation, are treated by the cryptographic protocol as integers in $\mathbb{Z}_p$. This is, however, an incorrect assumption as the encoded numbers actually represent fractional values and not integers. This can pose problems when arithmetic is performed on the encoded values, as they may not be meaningful for the representation that the numbers are in. We must therefore carefully consider the arithmetic operations used, specifically addition and multiplication, under this encoding. To do this, we examine how each operation treats the encoded value as if it were encoded as an integer in two's complement, and see what the effect this has on the encoded value.

For addition, the effect is the same for both integer and fixed-point representation. This is because the fixed-point representation can be seen as an integer that is shifted by the number of decimals as a power of 2. For example:

Bits		Integer		Fixed-point	Factor
0010 1110	=	46	=	2.875	$\times 2^4$

are equivalent. Addition therefore works correctly for both representations.

Bits	Integer	Fixed-point	
0010 1110	46	2.875	
0010 1010	42	2.625	+
0101 1000	88	5.5	

However, multiplication gives a different result in both representations. This is precisely because of the shift by the power of 2.

Bits	Integer	Fixed-point reality	F-P expectation	
0010 1110	46	2.875		
0010 1010	42	2.625		$\times$
0111 1000 1100	1932	$120.75 = 7.546875 \times 2^4$	7.546875	

After multiplication, the value contains an additional factor of 2^4. This may seem problematic, but we can mitigate this by paying careful attention to where we exactly perform these multiplications. It is impossible to perform multiplications in the encrypted format, as our commitment scheme is only additively homomorphic. Particularly in the L_2-norm proof, we only perform multiplication in $\mathbb{Z}_p$ once. Specifically, when we commit to the square of x, we perform $x \cdot x$ for each parameter. The squares are then summed to compute the squared L_2-norm of the update. Therefore, the squared L_2-norm of the update is shifted by 2^m where m is the number of decimals in the fixed-point encoding. We can overcome this issue by also shifting the range R by 2^m. Concretely, this means that instead of providing the L_2-norm range proof for the range $[0, R]$, but for the range $[0, 2^m R]$

5.5.2 Arbitrary Ranges

Supporting arbitrary ranges improves the usability of the protocol and provides a clear example of the aforementioned efficiency that the L_2-norm has in comparison to the L_∞-norm. In the previous construction of the protocol, the Bulletproof range proofs that are used can prove that a value lies within a range $[0, 2^n]$. However, it might be the case that additional flexibility is required, beyond bounds that are a power of 2. As shown in Chapter 4, the norm bound could affect accuracy if chosen imprecisely. Camenisch et al. [29], show how to construct a range proof for an arbitrary range $[A, B]$, from two range proofs that work with ranges of the form $[0, u^\ell]$ for some integers $u, \ell > 0$, which is the case for the Bulletproofs.

We can use this construction in our protocol to enforce arbitrary norm bounds. In the case of the L_∞-norm, enforcing arbitrary ranges requires a significant additional overhead. This is because for every parameter, we need to create and verify an extra range proof, which adds a factor of $O(n)$ to the protocol where n is the number of parameters. Non-asymptotically speaking, the per-parameter range proofs take up a large part of the computation time, so doubling this factor would contribute significantly to the total overhead.

In contrast, for the L_2-norm proof, we only need to require a single extra range proof on the squared L_2-norm commitment. Because we only add a constant number of range proofs, specifically one, the additional complexity of this is negligible.

5.5.3 Selecting a Norm Bound

In our secure aggregation protocol we incorporate norm bounding to contain the impact of malicious inputs. However, the norm bound must be chosen carefully so that it defends against model poisoning attacks while not harming global accuracy. In Chapter 4, we look at the implication of the norm bound on model accuracy and effectiveness of defending against the attacks. Additionally, from the analysis in Section 4.6.4 we provide a technique to find a suitable norm bound by simulating a typical client given the public training parameters. In this section, we show how the server can apply this technique in our protocol under secure aggregation to select a suitable norm bound. We suggest two methods for selecting the bound: Client simulation and Voting.

Client Simulation: The server performs several iterations of model training offline, using the exact training configuration as used in the network. The server uses the output of these simulations as the distribution of client updates to select the norm bound with, following the description in Section 4.6.4. For this method, the server needs to have access to some training data for the task. This is a reasonable assumption, because training data is often already used in federated learning by the server to calibrate the hyperparameters used in the machine learning training process.

Voting: The server initiates a voting round in which it selects a group of clients that create a regular update on the current model. The clients then calculate the statistic of their update, either the L_∞-norm or the L_2-norm, and engages in a secure voting protocol. The clients cast a vote on a bin in a range of bins, each bin representing a discrete interval with the center representing a value for the norm bound. Given the votes on the bins and α, the maximum fraction of adversarial clients in the network, the server ignores the α fraction of votes that were cast in the highest intervals. This is because the adversary could attempt to influence the norm bound selection process by voting for arbitrarily high intervals. The rest of the votes represent a distribution of client update sizes, of which the server can pick its norm bound as described in Section 4.6.4.

Both methods are suitable under different conditions: Voting requires the execution of an additional secure voting round, which has additional communication overhead. Conversely, for Client Simulation, the availability of training data that accurately represents the data distributions at the clients is required. Additionally, as update sizes tend to decrease with model convergence, both procedures can be applied repeatedly during the training process to re-calibrate to select the norm bound that is fitting for the period.

5.6 Optimizing for Scalability

The previously described protocol introduces significant computation and bandwidth overhead to ensure robustness against model poisoning attacks. In absolute terms, the computational overhead is significant because of the per-parameter zero-knowledge range proofs. Additionally, the costs of the protocol scale roughly linearly in the number of parameters, as shown in Table 5.1. This extra overhead is unworkable for federated learning and prevents the protocol from being used in practice because the costs of the protocol become unmanageable for a number of parameters that is reasonable in deep learning models. We hence pose the following research question:

> **How can the overhead of the protocol be reduced to support deep-learning models in practice?**

One can take two paths to attempt to answer this question. The first direction is to work on better optimized and efficient cryptographic constructions for secure aggregation and zero-knowledge proofs. The second direction is to work on reducing overhead by relying on protocol optimizations and compression techniques. In this work, we take the second direction and explore two classes of optimizations: Compression techniques from machine learning and protocol improvements, which we now further discuss.

	Complexity
Bandwidth (per client)	
Upload Commitments	$O(D)$
Upload Range Proofs	$O(\log(D)\log(n))$
Upload Rand Proofs	$O(D)$
Download Model	$O(D)$
Client computation	
Range Proofs	$O(Dn)$
Rand Proofs	$O(D)$
Training	$O(D)$?
Server computation	
Range Proofs	$O(Dnk)$
Rand Proofs	$O(Dk)$
Aggregation	$O(Dk)$
Discrete Log	$O(D)$

Table 5.1: Asymptotic complexity of our protocol in terms of bandwidth, client computation, and server computation, expressed in the number of parameters D, number of bits n in range $[0, 2^n]$, and number of clients k.

The first class of optimizations are focused on reducing the amount of information required to be transmitted in the model updates. For instance, by lowering the number of parameters necessary for the task, the number of cryptographic operations we have to execute is directly reduced. Much work in this direction exists, coming from the machine learning field. However, applying optimizations from machine learning is highly nontrivial because of two reasons. First, optimizations have to be already applicable at training time. For instance, model compression techniques that are applied after training, such as weight pruning [56, 92] and model distillation [60] are not useful for our setting. Second, compatibility with our protocol can be an issue due to the use of secure aggregation and the norm bounding that we want to enforce on top of the updates. Optimizations from machine learning must therefore be compatible with secure aggregation and preserve the meaningfulness of the L_∞- or L_2-norm bound.

The second class of optimizations attempts to reduce the costs of the protocol itself, for instance, by reducing the number of zero-knowledge proofs required per client. We now discuss the optimizations that we use in our protocol.

5.6.1 Probabilistic Quantization

A first optimization that reduces the amount of information required per parameter is probabilistic quantization. Probabilistic quantization has been shown to be a particularly efficient method to compress parameters in client updates from many clients [68]. Parameters are reduced to an encoding in a lower amount of bits using a probabilistic transformation. For a more detailed description, we refer to Section 3.3.1.

We can apply probabilistic quantization for the compression of both the model uploads from the clients to the server and the model downloads from the server to the clients. However, from theory we know that probabilistic quantization for model downloads is probably not feasible as it will impact model accuracy [27]. This is because we need to average over several updates in order for the compressed value to be a good estimator for the original value.

In our protocol, using probabilistic quantization to reduce the size of the parameters will not bring any gain in terms of bandwidth. This is because the parameters are encrypted before being uploaded to the server; hence, they are represented as exactly one group element, regardless of encoding.

Probabilistic quantization does give us an improvement for the range proofs in terms of computation time and bandwidth. Recall that the computational complexity to generate and verify range proofs is linear in the number of bits in the range. Additionally, for bandwidth the proof size grows logarithmically in the number of bits in the range due to the recursive inner-product argument that Bulletproofs use. With probabilistic quantization, we can reduce the number of bits required to express the range, which linearly reduces proof generation time and proof verification, and logarithmically reduces the amount of bandwidth required. In addition to overhead reduction, probabilistic quantization may also improve accuracy in comparison to deterministic quantization. [74].

5.6.2 Federated Dropout

Federated dropout can be used to reduce the number of parameters that are transferred in the model, while simultaneously improving the generalization effect of the model by providing a kind of weight regularization. The server picks a *dropout rate* r with $0 < r < 1$, which is a fraction of weights it will randomly withhold for each client. As a consequence, every client receives a different sub-model of the main global model. After training, the server aggregates the weights of the clients back into the global model.

Concretely, federated dropout provides a reduction of rD parameters in each model. As every component in the protocol is dependent on D, this results in significant reduction in overhead for computation and bandwidth. Only the Discrete Log operation must still be performed for D parameters, as the global model still has D parameters.

While federated dropout can cause a significant reduction in overhead for all the components by reducing the number of parameters that sent to each client, the impact is limited because a dropout rate r of less than 0.5 is unrealistic in practice for most models and tasks. At this extreme, a dropout rate of 0.5 would cause a 2x reduction in overhead.

5.6.3 Random Mask

A third improvement to reduce the size of the model update is random mask. In random mask, the clients do local training by only optimizing on the weights that are contained in a random mask unique per client, and only transmit these parameters to the server. The random mask is generated using a random seed that is agreed upon between the server and each client. It has been shown that a random mask containing only 12.5% of the original number of parameters does not significantly reduce accuracy [68]. A more detailed explanation of how random mask works can be found in Section 3.3.2.

However, there does exist a security consideration with this optimization method. Namely, the seed for the random mask must be generated in such a way that it is truly random, verifiable by the server and client. Otherwise, the server could manipulate the client into being the only contributor for a subset of model parameters, potentially revealing private information. Conversely, an adversary could freely choose which parameters it wants to contribute to, whereas the benign clients contribute at random, giving the adversary a bigger influence. In order to overcome this problem, the server and client agree upon a random seed using Diffie-Hellman key agreement.

5.6.4 Random Subspace Learning

A fourth optimization that reduces the amount of information transmitted in the model update is random subspace learning. This technique, initially introduced in the context of machine learning to understand the hardness of machine learning tasks [75], is useful to our protocol. This is because it reduces the number of parameters required for a model, while still preserving the L_2-norm in the transformation, ensuring compatibility with our L_2-norm bound restriction. Hence, we explore this technique further and are the first to do so in the context of federated learning. We now explain how random subspace learning can be applied specifically in our protocol, for a full explanation of random subspace learning we refer the reader to Section 3.3.3.

In random subspace learning, the model is trained in a random subspace θ^d that has a lower dimension d than the original model's dimension D. This reduces the number of parameters to be encrypted from D to d. The random subspace θ^d is projected onto the original model space θ^D using an orthonormal projection matrix $P \in \mathbb{R}^{D \times d}$:

$$\theta^D = \theta_0^D + \mathrm{P}\theta^d$$

This improvement is compatible with the L_2-norm bound because the orthonormal transformation preserves the L_2-norm. During training, θ_0^D and P are treated as constants, and optimizations are done on θ^d. Therefore, only θ^d has to be exchanged between the client and server in each round, resulting in an improvement in the number of parameters of a factor $\frac{d}{D}$.

Empirical data from the original authors has shown that for some tasks and model architectures a reduction of a factor 100 is possible [75]. However, there exists a tradeoff between the reduction factor $\frac{d}{D}$ and accuracy. The authors deem an accuracy of 90% of the original baseline acceptable, which may not always be the case. Therefore, while the optimization seems very promising for federated learning, we must carefully inspect the effect of random subspace training on accuracy.

5.6.5 Probabilistic Checking

An improvement to the protocol itself is probabilistic checking. In the current protocol, clients generate a range proof for every parameter, which is expensive in terms of computation. To reduce the amount of range proofs that are required, the server may choose to check range proofs probabilistically. At first sight, it may seem that this reduces security significantly. However, it turns out that the probability for a client to cheat is very low.

To implement probabilistic checking, the protocol is changed so that the clients first submit the vector of commitments to all the parameters. Then, the server chooses selects a fraction of parameters at random that it wants to receive a range proof for. Only after this, the clients respond with range proofs for these specific parameters. The selection of the parameters can be made efficiently by sending a random seed that is used as input to a PRG to deterministically generate the indices of the parameters in the vector.

To see why this works, assume the server chooses a fraction p of parameters uniformly at random. After receiving the request for the range proofs, the clients have to send the range proofs for the parameters that were sent in the commitments, due to the binding property of the commitments. The only way of cheating is to hope that the server does not choose the specific parameter that is out of bounds. For single parameter, this probability is $\frac{1}{p}$. However, in order to perform a reasonable model poisoning attack, the adversary has to use a significant portion of the weights.

The probability of detecting a cheating client follows a hypergeometric distribution. This implies that even with a small amount of malicious parameter, the probability of detecting the client quickly approaches 1, as shown in Figure 5.4.

Unfortunately, this optimization is only applicable for the L_∞-norm bound. For the L_2-norm bound, the adversary can compromise the computation with only a single parameter that is used to overflow the L_2-norm calculation. The adversary can then submit a large update that still satisfies the L_2-norm, due to the arithmetic overflow in the group. Probabilistic checking therefore does not work, because the attacker can perform this attack with a detection rate of $\frac{1}{p}$.

5.6.6 Optimistic starting

We now discuss a second optimization that works at the protocol level on the server-side. It is related to when norm bound proof generation and verification are being done, which take up most of the time in the protocol. The optimization is based on the insight that when the clients generate the norm bound proofs, the server is

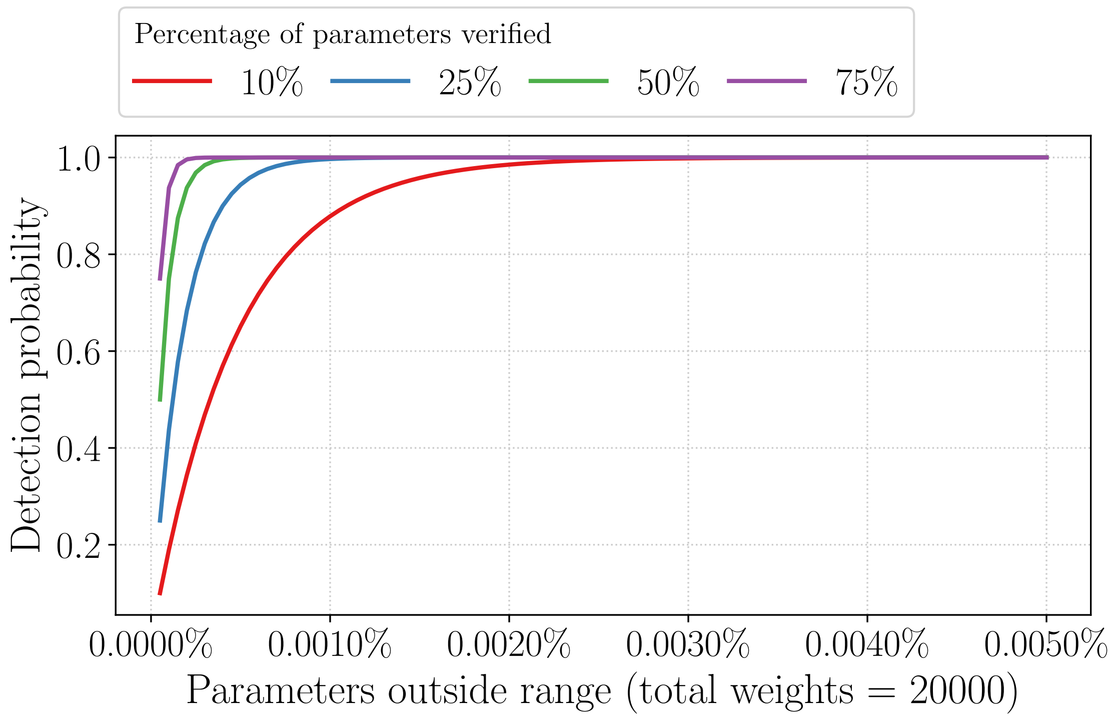


Figure 5.4: Example of the detection probability

waiting idly. Conversely, the clients are waiting for the server to verify the norm bound proofs to start the next round.

We can improve the time per round drastically by executing both operations in parallel. The server, upon receiving the update, optimistically attempts to decrypt the update and starts the next round. Only after this, the server verifies the norm bound proofs and aborts the new round if there are any inconsistencies.

This optimization allows the proof generation and verification to be done in parallel and does not compromise security. The only thing an adversary could gain is that a set of clients starts training the next round, which has to be discarded in addition to the previous round, which already had to be discarded without this optimization. However, as explained in Section 5.1.1, this Denial-of-Service attack is detectable and we therefore deem it out of scope of our threat model.

6 Implementation

In this chapter, we discuss the prototype implementation of our Secure Federated Learning Protocol described in Chapter 5.

This implementation is an extension of the implementation by Lei et al. [74]. We thus only describe the general architecture and the additions that were made. For a detailed description of the implementation of unaltered components such as the Foreign Function Interface or the Baby-Step-Giant-Step algorithm, we refer to their work [74].

6.1 Overview

The Secure Federated Learning Framework is implemented to perform federated learning training across multiple nodes, while aggregating updates using secure aggregation and enforcing norm bounds using zero-knowledge proofs. In this section, we give an overview of the framework.

The system consists of three components: The server, the client, and the cryptographic library. The server handles the coordination of the federated learning protocol and communicates with the clients. The client implements local training. The cryptographic library implements the cryptographic operations used for secure aggregation and zero-knowledge proofs on which both the server and client rely.

The server and the client are built using Python, due to its ability for fast prototyping and its wide adoption in the machine learning community through many machine learning frameworks. The machine learning framework of choice is Tensorflow [7], programmed through the Keras interface [32]. The advantage of Keras is that in theory two backends in addition to Tensorflow are also supported, namely CNTK and Theano. Furthermore, Keras allows for the serialization of model structure and parameters, required for transferring the model between the server and the client.

The server and client communicate through the Socket.IO protocol [91] over the network. Furthermore, they interact with a local instance of the cryptographic library using a Foreign Function Interface. An FFI allows programs compiled from one language to call functions from a program written in another language. The cryptographic library generates and verifies the cryptographic objects needed for secure aggregation and the zero-knowledge proofs. These objects are serialized through the FFI, sent by the client to the server over the network, and then deserialized by the server's FFI with the cryptographic library. A schematic overview is shown in Figure 6.1.

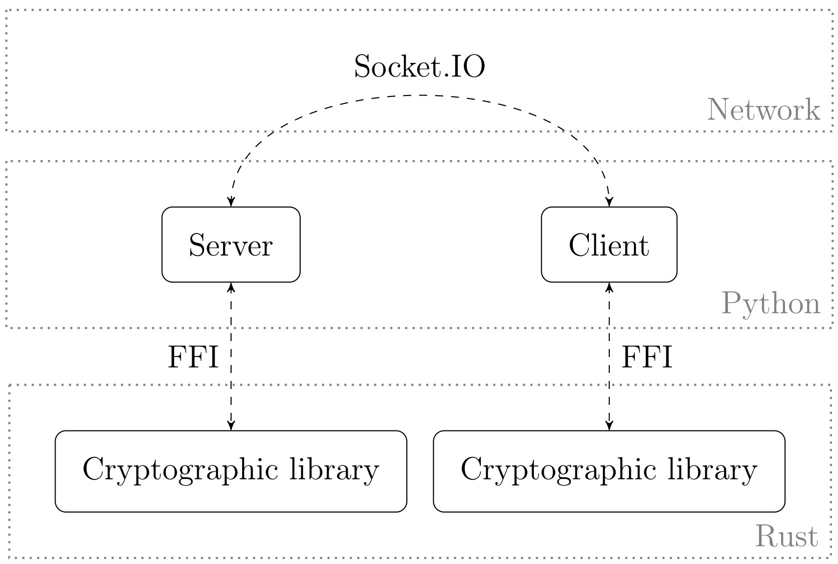


Figure 6.1: High-level architecture of the system.

Message flow The clients and server communicate through a series of asynchronous protocol messages which roughly correspond to the protocol steps, shown in Figure 6.2. We now describe this flow of protocol messages in the framework. For a full explanation of the protocol, we refer the reader to Section 5.2.

First, the clients establish a connection with the server and register using `client_register` to participate in the federated learning setup. Then, the server transfers the machine learning and security configurations using `transfer_model_config` and `transfer_crypto_config`, respectively, as well as the model definition and the random initialization of the untrainable weights using `transfer_model`. Because the number of untrainable weights can be very large, such as in random subspace machine learning, the implementation supports the transfer of a seed to generate the random initialization instead, using `transfer_seed`.

When the server has received enough client registrations as defined by the configuration, it starts a training round by transferring the current global model weights `start_training`. Note that in this step, only the trainable weights are transmitted. After a client is finished training, they send the commitments to the server with the training `training_finished` message. For probabilistic checking of range proofs, the server then selects the commitments it wants to see a range proofs for with `select_commitments`. Finally, the client returns the requested range proofs in the `transfer_range_proofs` step.

6.2 Aggregation Protocol

After having established a general overview of the framework's structure, we now discuss the different kinds of aggregation that the framework supports, and how this is implemented. The most important feature of the framework is the support for

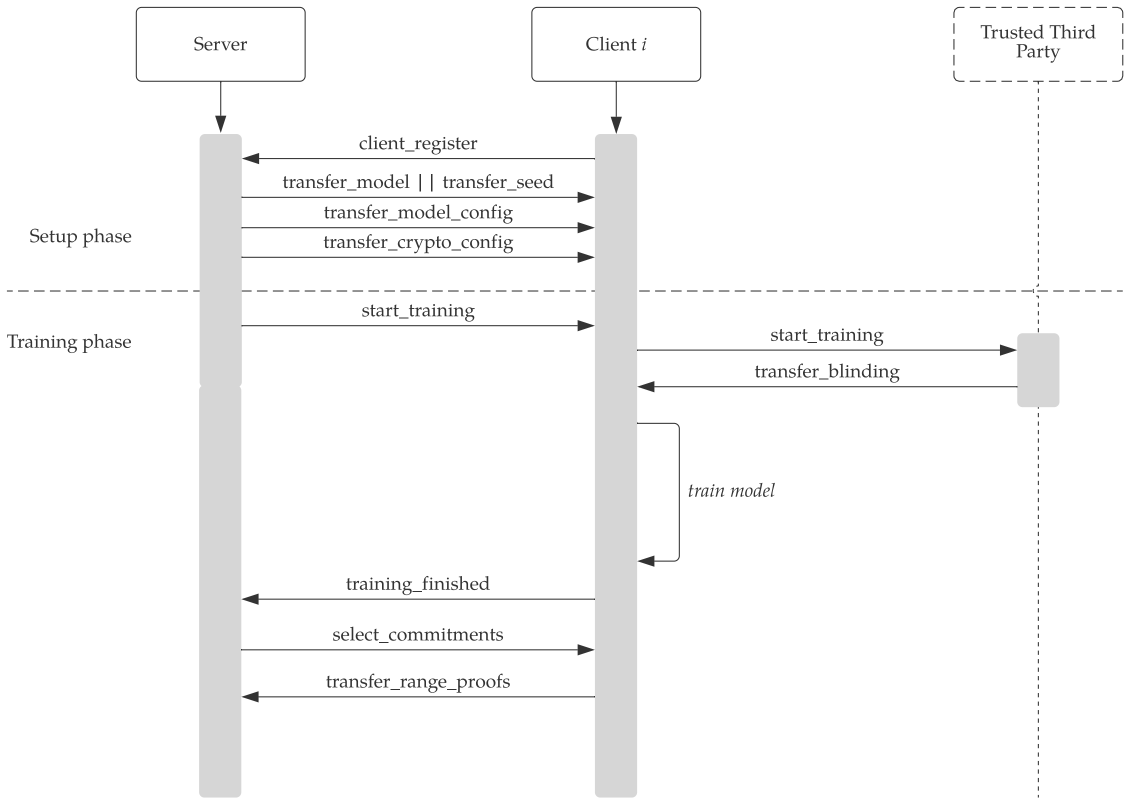


Figure 6.2: Protocol message flow.

secure aggregation with norm bounding. In addition to this, the framework supports other aggregation protocols in order to compare the aggregation protocols. This functionality is captured in a set of *aggregators*. The aggregators vary in level of security and performance and operate as interchangeable modules in the system that can be easily changed without having to change the underlying federated learning system. The following aggregators are supported:

1. `plain_aggregator` This component does not use any encryption or robustness and sends the updates in plain text to the server. This aggregator serves as the baseline for benchmarks.

2. `secure_aggregator` This component aggregates using only the secure aggregation part of the protocol, without any robustness proofs.

3. `range_aggregator` In addition to the functionality of `secure_aggregator`, this component adds parameter-wise range proofs and randomness proofs to prove the L_∞-norm bound.

4. `l2_aggregator` In addition to the functionality of `secure_aggregator`, this component adds the robustness proofs required for the L_2-norm bound.

The various aggregators can be used interchangeably, depending on the specification in a configuration file. This allows for easy comparison and benchmarking of the different aggregation methods.

6.3 Cryptographic Library

The aforementioned aggregators rely heavily on cryptographic operations for their functionality and security guarantees. The cryptographic functionality is implemented in a separate library, for efficiency and maintainability reasons. The cryptographic library handles all of the operations required for secure aggregation and zero-knowledge proof generation and verification. We begin this section by describing the implementation of the cryptographic primitives, and then go on to discuss the implementation of the zero-knowledge range and squared randomness proofs.

We use the algebraic structure of an elliptic curve over a finite field as the instantiation of the group $\mathbb{G}$ for our cryptographic operations. Concretely, we use the standardized curve Curve25519 which is specifically designed to be fast and immune against timing attacks and provides 128 bits of security. Furthermore, the Bulletproofs library [37] also uses this curve, making compatibility easier.

The architecture of the cryptographic library is based on the primitives of the curve25519-dalek library [64]. The library represents elements from the group $\mathbb{G}$ with prime order p as 256-bit `RistrettoPoint` structures. Additionally, elements in $\mathbb{Z}_p$ are represented as 256-bit `Scalar` structures. For the generators g and h, we follow the standards of the library to ensure they are chosen safely. Specifically, g is set as the base point of Curve25519, i.e. for $x = 9$, and h is set to the `SHA3_512` hash of g. We assume that the relation between g and h is not known, which is imperative for security. For this we need to believe the `SHA3_512` behaves as a random oracle.

Number representation The cryptographic library performs the conversion process from floating-point values to fixed-point and then to `Scalar` values. The size of the fixed-point representation can be configured using configuration parameters ρ and κ, representing the total number of bits and the number of fractional bits in the representation, respectively. After converting the floating-point numbers to fixed-point using deterministic or probabilistic quantization, the ρ-bit numbers have to be converted to 256-bit `Scalar`s. However, we have to ensure that the representation of the numbers in `Scalar` form correspond to the correct fixed-point values, so that operations done on `Scalar` values correctly translate. We now explain how this conversion process works and how it ensures the correct representation.

As $\rho \ll 256$, the positive fixed-point numbers can be embedded directly into the least-significant bits of the 256-bit `Scalar`. However, this does not work for negative values, as the most significant bit in the fixed-point representation that stands for the sign does not translate to the most significant bit in the `Scalar` representation. Concretely, this results in the problem that the inverse $-p$ of a number p is not the same in the `Scalar` representation. In order to solve this, we first convert the negative number $-p$ to its positive representation p, convert it to its `Scalar` representation P and then negate the `Scalar`, $-P$.

6.3.1 Zero-Knowledge Range Proof

The implementation of the Bulletproof range proofs is done using the *Bulletproofs* library by the Dalek Cryptography Developers [37]. This library also relies on the same primitives from the curve25519-dalek library. The library supports fast range proof generation and verification using the AVX2 instruction set [90].

Flexible ranges A limitation of the library is that only range proofs for 8-, 16- and 32-bit ranges are supported. Because the bounds that we would like to enforce are more subtle than bytes, we want to construct a range proof for a more flexible range. To achieve this, there are two approaches, shifting the decimal point or adding extra range proofs.

A first solution is to adapt the fixed-point configuration to this limitation, by shifting the decimal point in the fixed-point representation such that the size is exactly one of the three supported ranges. Consider, for example, a fixed-point representation of 12 bits with 10 fractional bits. If we require a norm bound of 4, we can shift the decimal point up 4 bits, resulting in a fixed-point representation of 16 bits with 14 fractional bits. There are two issues with this solution. First, it forces us to increase the number of bits required per parameter, which can incur additional overhead. Second, the solution still only supports norm bounds that are a power of two as visualized in Figure 6.3, which may not be flexible enough for our purpose.

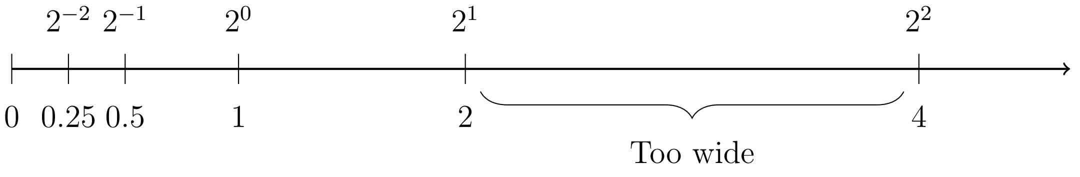


Figure 6.3: Visualization of the norm bounds that are powers of two.

Therefore, the second solution is based on adding an additional range proof to enforce an arbitrary range $[A, B]$, as explained in Section 5.5.2. The implementation for this solution is different for both norm bounds. For the L_∞-norm bound, every parameter-wise range proof gets a second range proof. This has significant cost, as the range proofs are already a large part of the protocol overhead. For the L_2-norm bound, arbitrary ranges are much more efficient to implement. Only the range of the squared L_2-norm requires an additional proof, which is only a small and constant additional factor.

6.3.2 Zero-Knowledge Squared Randomness Proof

We implement the zero-knowledge proofs to prove the randomness and the square relation following the specification given in Section 5.4.3. The protocols are implemented using Merlin transcripts [36]. Merlin abstracts the Fiat-Shamir heuristic away so that non-interactive proofs can be implemented as if they were interactive proofs, which improves readability and reduces the risk of errors. The generation and verification of the squared randomness proofs are implemented as follows.

The `create_squarerandproof` FFI function takes a vector of `float32` input values and two vectors of `Scalar` blinding values as input, to generate the ElGamal commitments and the Pedersen commitments to the square. The function returns a vector of squared randomness proof transcripts and a vector of `SquareRandProofCommitment`, which each contain a tuple with an ElGamal commitment and a Pedersen commitment to the square. The `verify_squarerandproof` FFI function takes the vector of squared randomness proofs and the vector of commitments and returns a single boolean indicating whether the proofs are valid.

6.3.3 L_2-norm proof

As explained in Section 5.4.2 L_2-norm proof has two components that perform operations the same set of commitments. Implementing each component separately requires multiple exchanges between Python and the cryptographic library through the FFI, resulting in additional overhead because of the serialization and deserialization of the parameters. Moreover, having multiple interactions between the two components increases the complexity of the code. Thus, to reduce the overhead and to aid the simplicity of the Foreign Function Interface between the Python and the Rust code, we created two helper functions to perform the additional steps required to generate and verify the L_2-norm bound proof. These helper functions are called in addition to the parameter-wise range proofs.

Similar to `create_squarerandproof`, the `create_l2proof` FFI function takes a vector of `float32` input values and two vectors of `Scalar` blinding values as input, as well as the required L_2-norm. Additionally, it requires a parameter to indicate how many cores the range proof generation should be split over. The function then computes the squared randomness proof, adds the squared commitments together to get the squared L_2-norm, and computes range proof for the squared L_2-norm. It returns the commitments, the squared randomness proofs, and the single range proof of the squared L_2-norm. The `verify_l2proof` FFI function takes the squared randomness proofs, the commitments, and the single range proof for the squared L_2-norm as input. It then verifies the squared randomness proofs, computes the squared L_2-norm from the commitments, and verifies that the range proof is correct and is for the same squared L_2-norm commitment.

These two helper functions, in combination with the parameter-wise range proofs, are sufficient to generate and verify the L_2-norm bound proofs.

7 Evaluation

In this chapter, we evaluate the Secure Federated Learning Framework. The objective of the evaluation is to answer the following questions: (1) What is the overhead of the secure federated learning protocol in terms of computation, bandwidth and accuracy loss, (2) how much can the proposed optimizations reduce this overhead, and (3) can we apply the protocol in practice?

This chapter is structured as follows: We first define the experimental setup in which of the evaluation After that, we look at the performance overhead of the cryptographic operations. Finally, we show an end-to-end benchmark of the full system, highlighting the overhead reduction of the optimizations.

7.1 Experimental Setup

In this section, we describe the methodology and the configurations used in the experiments.

7.1.1 Methodology

The evaluation quantifies overhead of the protocol for the following metrics:

1. **Cryptographic computation time:** The time required to perform the cryptographic operations
2. **Bandwidth:** The amount of data transmitted by the protocol.
3. **End-to-end time:** The time it takes to do a full run of the end-to-end system.
4. **Accuracy:** The accuracy of a run of the end-to-end system.

We perform microbenchmarks of the cryptographic library using the cargo bench module [1]. For each statistic, we take the average of 40 measurements after 'warming up' the processor by doing one mock execution.

7.1.2 Setup

The micro and end-to-end benchmarks are performed on different setups.

Microbenchmarks: We use a single AWS EC2 *c5d.4xlarge* instance. The machine contains 32 GB of memory and 16 cores of a first-generation Intel Xeon Platinum 8000 processor that supports AVX2 instructions.

End-to-End benchmarks: We use five AWS EC2 *c5d.9xlarge* instances. The machines each contain 64 GB of memory and 32 cores of a first-generation Intel Xeon Platinum 8000 processor that has support for AVX2 instructions. The server occupies one instance, while the clients are evenly distributed over the other four instances. The server and clients communicate inside an AWS Virtual Private Cloud.

7.1.3 End-to-End Configuration

We evaluate the end-to-end system under several tasks, models, and robustness configurations. Furthermore, we compare the performance of the protocol for the MNIST and CIFAR-10 tasks. For both tasks, we use specific a Convolutional Neural Network (CNN) of a size that is suitable for the task. Both CNNs follow the same architecture: two convolutional layers, a 2 by 2 max-pooling operation, and two fully connected layers.

For the MNIST task, we use a model with 19166 parameters. The convolutional layers consist of 8 and 4 channels, respectively, and the two fully connected layers contain 576 and 32 parameters, respectively. For the CIFAR-10 task, we use the LeNet5 CNN of 62006 parameters. This network contains an additional max-pooling layer after the first convolutional layer of 6 filters. Then, a convolutional layer with 16 filters is applied, followed by fully connected layers of 120 and 84 parameters.

7.2 Cryptography

To find out the computational overhead of the protocol, we evaluate the computation time of the individual cryptographic components in the protocol. We evaluate the computation time of two components of the library: the square randomness proof and the full norm bound proof, of which the square randomness proof is a sub-component.

7.2.1 Zero-Knowledge Square Randomness Proof

We measure the performance of the square randomness proof and compare it with the regular randomness proof. The randomness proof is used in the protocol to ensure that the randomness of the two components in the ElGamal commitments is the same. The square randomness proof is an extension of the randomness proof, which additionally proves that one commitment contains the square of another commitment, used as a component in the L_2-norm bound proof.

The computation time for the randomness proofs grows linearly with the number of parameters, as shown in Figure 7.1. The 32-bit results in Figure 7.1 are representative for other parameter bit-lengths, because of the immunity of Curve25519 against timing-attacks [74]. For 2^{15} parameters, the latency to generate the randomness poof is 2.10 seconds whereas the cost for the squared randomness proof is 3.40 seconds (1.62x). The cost to verify the randomness proofs shows a similar increase: 1.04

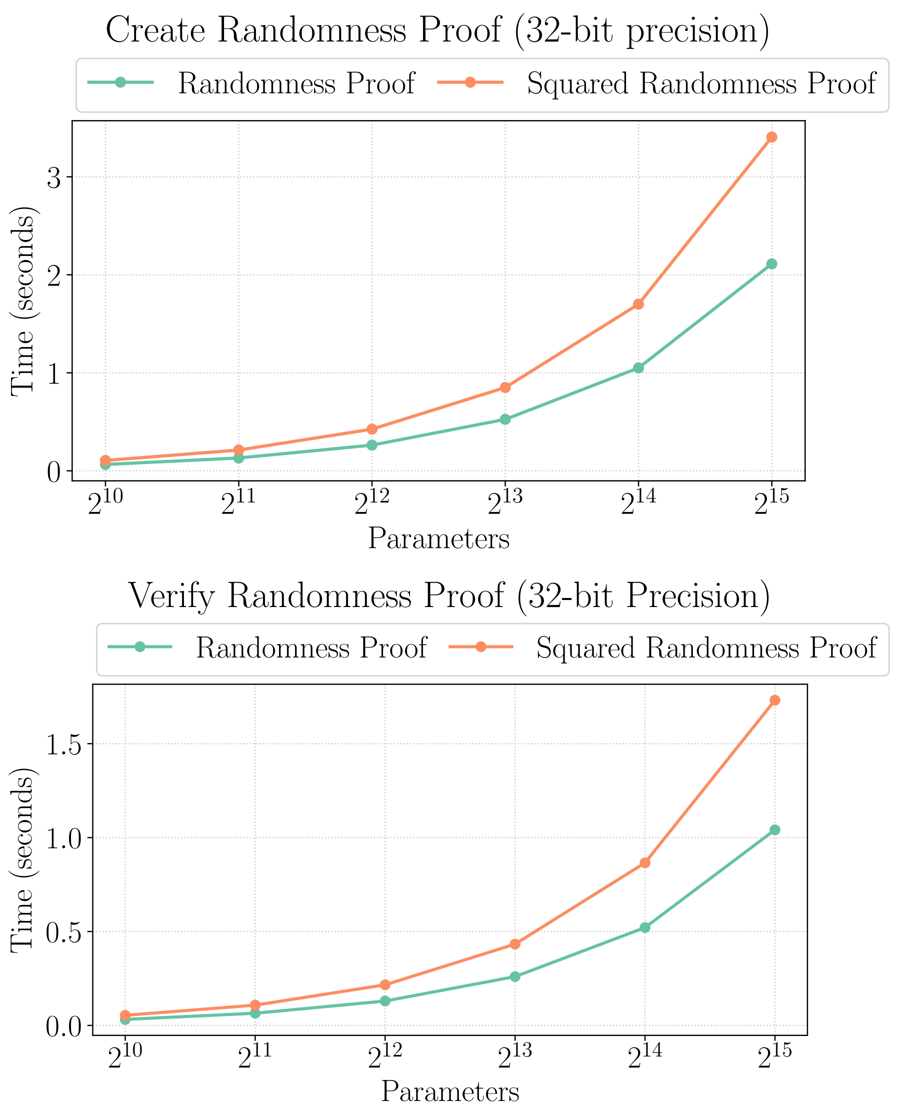


Figure 7.1: Performance comparisons of the squared randomness proof and the regular randomness proof under 32-bit representation.

seconds for the randomness proof and 1.73 seconds for the square randomness proof (1.67x). We attribute this to the fact that the proof of square relation protocol contains one additional constraint in comparison to the original randomness protocol, which already contains two. The prover and verifier are required to perform additional group operations to create and verify the extra Pedersen commitment, resulting in this extra computation overhead.

The additional overhead for the square randomness proof in comparison to the randomness proof is acceptable because the computation time of the randomness proof is only a small factor in the total norm bound proof computation time. Hence, we expect that this additional constant factor is relatively small with regards to the full norm bound protocol, which we evaluate in the next section.

7.2.2 Zero-Knowledge Norm Bound Proofs

We now evaluate at the performance of both the L_∞- and the L_2-norm bounds to quantify the computational overhead of the most computationally intensive parts of the protocol. The proofs are made up of different components, such as the per-parameter range proofs, randomness proofs, and optionally the L_2-norm range proof, as discuss in Section 5.4.2. We first show the comparison of both full proofs, and then highlight some advantages of the L_2-norm proof in terms of efficiency and flexibility over the L_∞-norm proof.

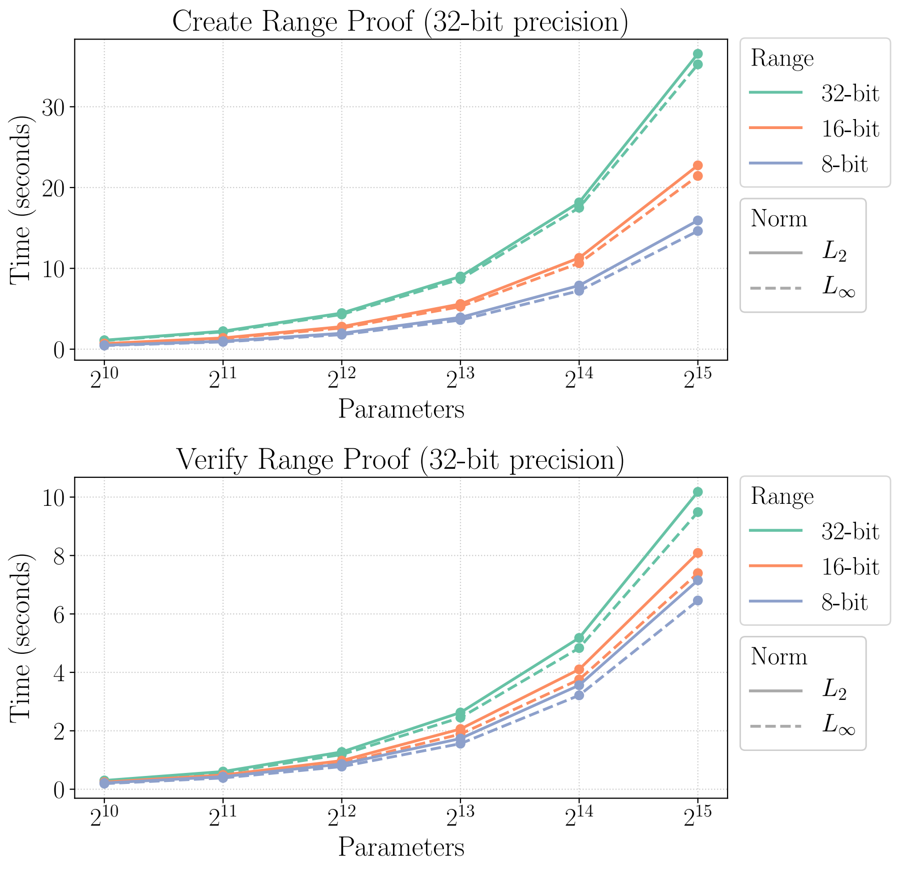


Figure 7.2: Performance comparisons of the full proofs for L_2- and L_∞-norm constraints under 32-bit representation.

Figure 7.2 shows that the time required to create and verify the norm bound proofs grow linearly in the number of parameters, and logarithmically in the number of bits in the proof range for the parameters. For 2^{15} parameters and a 32-bit bound, generating an L_∞-norm bound proof takes 35.24 seconds and 36.56 seconds for an L_2-norm bound proof (1.04x). For proof verification for 2^{15} and the same 32-bit bound, L_∞-norm bound proof takes 9.49 seconds and the L_2-norm bound proof takes 10.18 seconds (1.08x). The small additional overhead of the L_2-norm proof is in line with our expectation from the previous section: The squared randomness proof in the L_2-norm proof adds some overhead but turns out to be a very small factor (1.04x and 1.08x) in the actual L_2-norm proof.

The extra overhead of the L_2-norm bound is acceptable, because of two reasons. First, as opposed to the L_∞-norm bound, the L_2-norm bound is compatible with other optimization techniques such as random subspace learning, which could save more overhead than the additional cost of the L_2-norm in comparison to the L_∞-norm. Second, the L_2-norm proof allows us to be more flexible in the specific bound that we can enforce, which we discuss in the next paragraph.

Arbitrary ranges Because the L_2-norm bound itself is enforced by a single range proof on the sum of the squared commitments, we can choose to enforce a more costly, flexible bound for this single commitment. As explained in Section 5.5.2, the current Bulletproof implementation only supports proof ranges of 8, 16, and 32 bits. To bridge the gap to proof an arbitrary range $[a, b]$, an additional range proof is required. For both norms, the extra overhead incurred is different, because the L_∞-norm bound requires an additional range proof per parameter, whereas the L_2-norm bound only requires one additional range proof for the sum of squares commitment. Figure 7.3 shows that for 2^{15} parameters the generation of the L_∞-norm bound for an arbitrary range takes 68.37 seconds compared to 36.57 seconds for the L_2-norm bound (1.87x). Conversely, for verification the L_∞-norm bound takes 17.93 seconds compared to 10.18 seconds for the L_2-norm bound (1.76x), which is in line with our expectation.

7.3 Message Size

To answer the question of bandwidth overhead, we analytically look at the overhead of the size of the messages. We count the message size overhead as the number of bytes taken up by the elements sent between the server and the clients. Given the number of parameters D, we look at the message size overhead per client and per round.

Download Clients download the parameters of the model from the server in 32-bit floating-point format. Random subspace learning reduces the number of parameters downloaded by the clients, so given an intrinsic dimension d, the message size overhead of the model download bandwidth is reduced to $4d$ bytes.

Upload Upload message size overhead is a more interesting metric to look at, for two reasons. First, due to the use of cryptographic constructions, the uploading of the model update from the clients to the server is much larger than the model

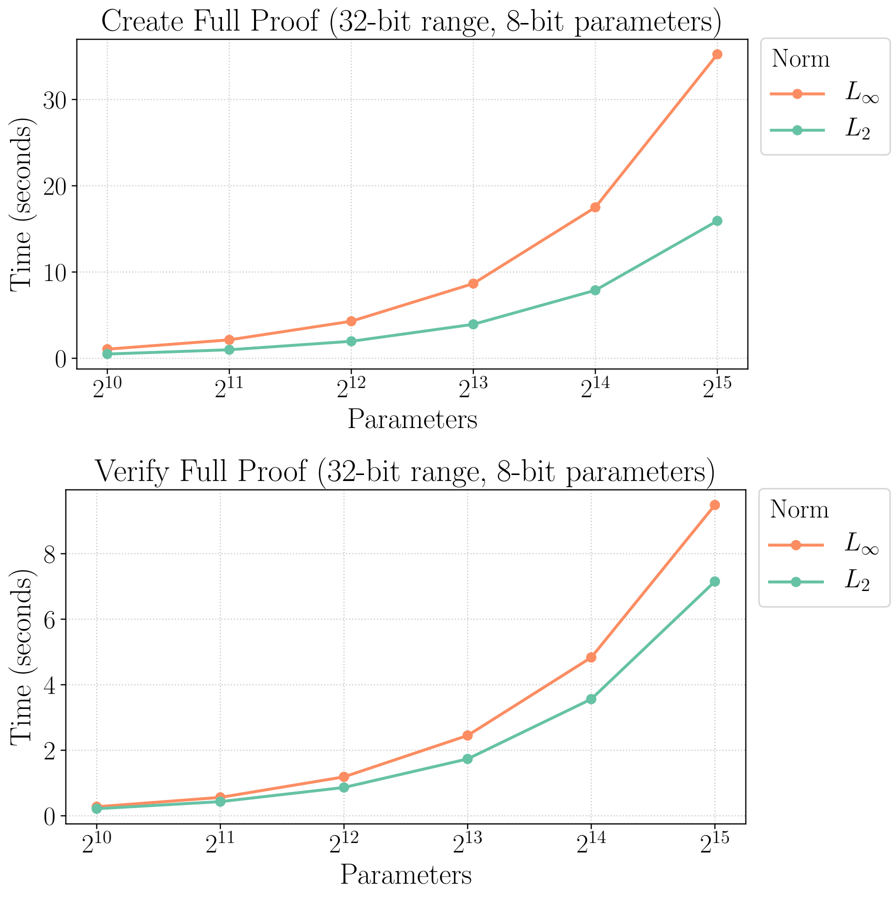


Figure 7.3: Comparison of L_2- and L_∞-norm bound proofs for an arbitrary range.

download. Second, client devices in a wide area network are often more constrained in terms of upload speed than of download speed.

We first analyze the message size for the individual components of which client update consist, and then show the totals for both norm bounds. Every `Scalar` and group element used in the cryptographic constructions consist of 32 bytes. A client update can consist of the following components, depending on the norm bound:

1. **Commitment vector:** The update is encrypted using ElGamal commitments, which each consist of two group elements, resulting in $64D$ bytes.

2. **Range proofs:** The per-parameter range proofs for a range n can be batched together to only take $2(\log_2(n) + \log_2(\texttt{next\_pow2}(D)) + 4$ group elements and 4 `Scalars` of 32 bytes, where `next_pow2` is rounds up the argument as the next power of 2. However, to distribute the heavy generation and verification operations over multiple CPU cores to execute these operations in parallel, we partition the vector into p segments. The total message size overhead then becomes $32p(\log_2(n) + \log_2(\texttt{next\_pow2}(\frac{D}{p})) + 9)$ bytes.

3. **Randomness proofs:** From Section 5.4.3, we know that per-parameter randomness proof to prove that the randomness in the ElGamal commitment tuple matches takes 2 `Scalars` and 2 group elements, totaling $128D$ bytes.

4. **Squared commitment vector:** For the L_2-norm bound proof, an additional vector of commitments to the squared parameters is required to compute the L_2-norm. These commitment are Pedersen commitments, which are represented by a single group element, as opposed to the ElGamal commitment vector. Therefore, they take up only $32D$ bytes.

5. **Square randomness proofs:** The squared randomness proofs, used to proof the square relation for the L_2-norm bound, are an extension of the randomness proofs and contain one extra `Scalar` and group element, resulting in $192D$ bytes.

6. **Squared L_2-norm range proof:** This range proof ensures that the L_2-norm of the commitment vector is within the given range n, and takes $32(\log_2(n)+9)$ bytes.

Norm bound	Message size overhead
L_∞ (1., 2., 3.)	$32(6D + p(\log_2(n) + \log_2(\texttt{next\_pow2}(\frac{D}{p})) + 9))$
L_2 (1., 4., 5., 6.)	$32(9D + p(\log_2(n) + \log_2(\texttt{next\_pow2}(\frac{D}{p})) + 9) + \log_2(n) + 9)$

Table 7.1: Overview of messages for both norm bounds, given the number of parameters D and the proving range n.

The total messages sizes for both norms are shown in Table 7.1, which reveal that the upload message overhead is independent of the number of bits in the parameter encoding. Only the size of the range in the range proofs is logarithmically in relation to the message overhead, because of the inner product argument used in the Bulletproof range proofs. The L_∞-norm bound proof consists of the commitment vector, the range proofs, and the randomness proofs. The L_2-norm bound proof consists of the commitment and squared commitment vector, the range proofs and the Squared L_2-norm range proof. The display of the relation of the message size to the number of parameters in Figure 7.4 shows that for large vectors, the commitments and the randomness proofs contribute most to the messages size. For instance, 2^{15} parameters under the L_∞-norm bound has a message size of approximately 6.3 Megabytes, of which the commitments and the randomness proofs take up 99.3%.

Nevertheless, we deem the message size overhead of our protocol acceptable because, while the increase in message size is significant for our protocol, in absolute terms the upload of several megabytes of data is not problematic. Especially because in federated learning, client devices are typically only selected when their connection to the internet is not limited [81]. Additionally, the optimizations discussed in

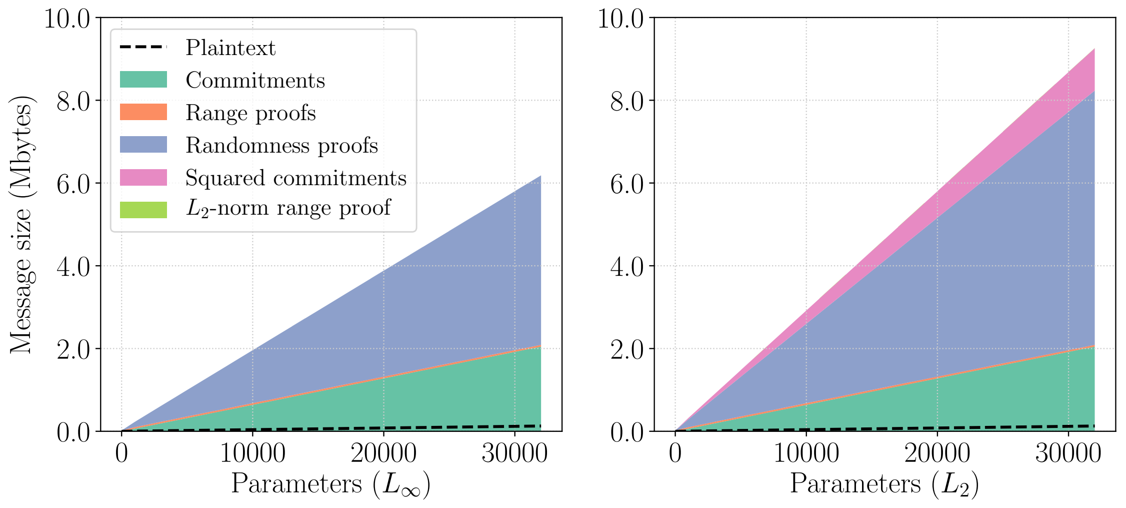


Figure 7.4: Message overhead per round for both norm bounds in the number of parameters D for a 32-bit range (n) and 64 range proof vector partitions (p)

Section 5.6 help reduce the message size overhead. For instance, random subspace learning reduces the number of effective parameters, directly corresponding to a lower message size overhead.

7.4 End-to-End Evaluation

To quantify the effects of optimizations on the protocol and to see if it is applicable for deep-learning in practice, we perform a full deployment of the prototype implementation of the protocol. We first look at the effect of the optimizations on the protocol. After that, we compare the results with an insecure, plaintext baseline.

7.4.1 Optimizations

We compare the optimized system with a baseline implementation with the norm bounds. For the baseline implementation, we use a (16,8)-fixed-point representation, similar to [74]. We use different optimizations for both norm bounds, because some optimizations are only applicable for a specific norm bound.

We evaluate the system under both norm bounds with the following optimizations:

Optimized L_∞-norm We apply probabilistic clipping with a rate of 10%, meaning the server will only request 10% of the range proofs after receiving the commitments.

Optimized L_2-norm We apply random subspace machine learning to reduce the amount of parameters that are trained in each round. For MNIST, we choose an intrinsic dimension of 4000 which gives us a 4.79x compared to the original amount of parameters. For the CIFAR-10 task, we used an intrinsic dimension of 10000, resulting in a 6.20x parameter reduction.

For both norm bounds, we apply probabilistic quantization to afford to reduce the number of bits per parameter to an (8,7)-fixed-point number representation. We additionally applied optimistic starting of the next round on the server-side.

L_∞-norm bound Table 7.2 shows that the optimizations provide a significant speedup for both norm bounds and tasks. For the CIFAR-10 task with 62006 parameters, the time per round is reduced from 660 seconds to 293 seconds (2.25x) for the optimized L_∞-norm bound with probabilistic clipping and optimistic starting. Moreover, Figure 7.6 shows that the use of the (8,7)-fixed-point representation does not reduce accuracy significantly for the L_∞-norm, which we attribute to the use of probabilistic quantization. In the MNIST task, the optimized L_∞-norm actually improves accuracy while a 2x smaller encoding is used. We attribute this the probabilistic quantization as well, which we further explore later in this section.

L_2-norm bound Coming back to the CIFAR-10 task, the L_2-norm bound optimization provides an even larger reduction than the L_∞-norm bound, from 802 seconds to 120 seconds (6.8x), as shown in Table 7.2. The optimized L_2-norm bound with random subspace learning is more than twice as fast as the optimized L_∞-norm bound, even though we saw in the previous section that the cryptographic operations take more time in the case of the L_2-norm. However, this additional speed comes at a price: in contrast to the L_∞-norm bound, the accuracy is slightly lower due to the limitations caused by the smaller intrinsic dimension from the random subspace learning, as shown in Figure 7.6. While there is some loss in accuracy, specifically 0.0105 and 0.0306 for the MNIST and CIFAR-10 tasks, respectively, this does not have to be problematic. On the contrary, the use of random subspace learning can be seen as a very efficient tuning mechanism to trade a small accuracy loss for a drastic performance increase by varying the intrinsic dimension.

Probabilistic quantization We attributed the accuracy improvement of the optimized norm bounds even though the number of bits per parameter was reduced to probabilistic quantization. However, we attempt to quantify the effects of probabilistic quantization by performing further experiments where the quantization is the only changing factor. Results in Figure 7.5 show the performance of the non-IID MNIST task for various fixed-point quantization methods, as well as a 32-bit floating-point baseline. The results show that probabilistic quantization to 8-bits with 7 fractional bits is feasible without reducing accuracy significantly compared to the baseline. Moreover, we can see that probabilistic quantization works significantly better than deterministic quantization: probabilistic quantization achieves roughly the same accuracy with half the amount of information. In our experiments, we only leave a single bit for the integer part in the fixed representation to leave the most space for the fractional part of the weights because machine learning parameter weights are typically between 0 and 1.

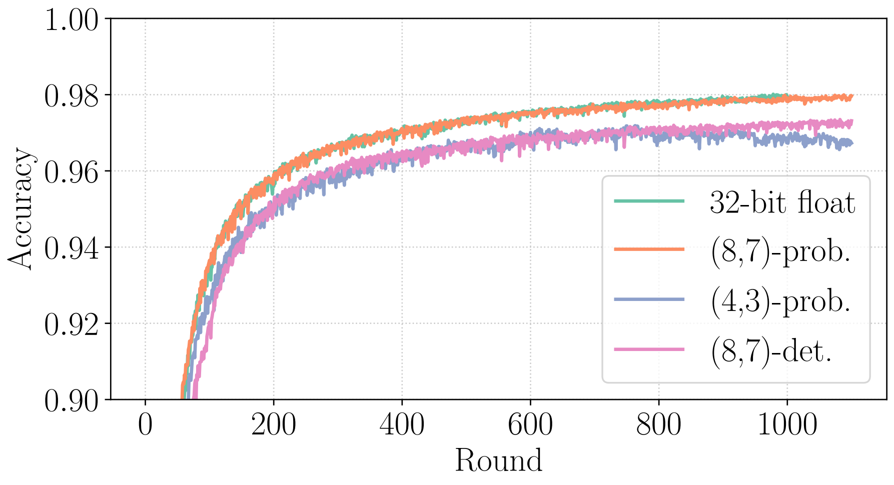


Figure 7.5: Comparison of quantization configurations. non-IID MNIST dataset, 3383 clients with 30 selected per round.

7.4.2 Plaintext Comparison

What is left is to see if our protocol is applicable in practice. To this end, we compare our optimized, secure norm bounding protocol with an insecure plaintext baseline. Table 7.2 shows that it takes 5.60 seconds for a model on MNIST task to train in plaintext, compared to 86.17 seconds (15x) for the L_∞-norm bound and 38.50 seconds (7x) for the L_2-norm bound. To train the model to a reasonable accuracy in 40 rounds, it takes 56 minutes for the L_∞ and 26 minutes for the L_2-norm, compared to 4 minutes for the plaintext baseline. The accuracy in Figure 7.6 of the L_∞-norm bound experiment shows only a slight reduction compared to the full 32-bit floating-point run, due to probabilistic quantization. Furthermore, for the L_2-norm we do see a slight reduction because of the use of random subspace learning, which we use as a tuning knob to trade accuracy for speed, as explained in the previous section.

For the larger CIFAR-10 model with 62006 parameters the time per round is 7.31 seconds for the plaintext baseline compared to 293.35 seconds (40x) for the L_∞-norm bound and 120.46 seconds (16x) for the L_2-norm bound. A full training session of 40 rounds translates to 196 minutes and 80 minutes for the L_∞- and L_2-norm bounds, respectively, compared to 5 minutes for the plaintext baseline. The round time of 5 minutes for the large CIFAR-10 CNN of 62006 parameters. Hence, we conclude that our protocol is applicable in practice for typical models used in deep learning.

Server load The examination of the current protocol bottlenecks shows that there exists a bottleneck at the server for update aggregation, decryption, and proof

		L_∞		L_2	
	Plain	Naïve	Optimized	Naïve	Optimized
Accuracy	0.9855	0.9815	0.9828	0.9815	0.9710
Time (s)	5.60 (1x)	278.45 (50x)	86.17 (15x)	335.77 (60x)	38.50 (7x)

(a) MNIST (19166 parameters)

		L_∞		L_2	
	Plain	Naïve	Optimized	Naïve	Optimized
Accuracy	0.5573	0.5612	0.5610	0.5612	0.5306
Time (s)	7.31 (1x)	660.35 (90x)	293.35 (40x)	801.80 (110x)	120.48 (16x)

(b) CIFAR-10 (62006 parameters)

Table 7.2: Final accuracy and time per round for both norm bounds.

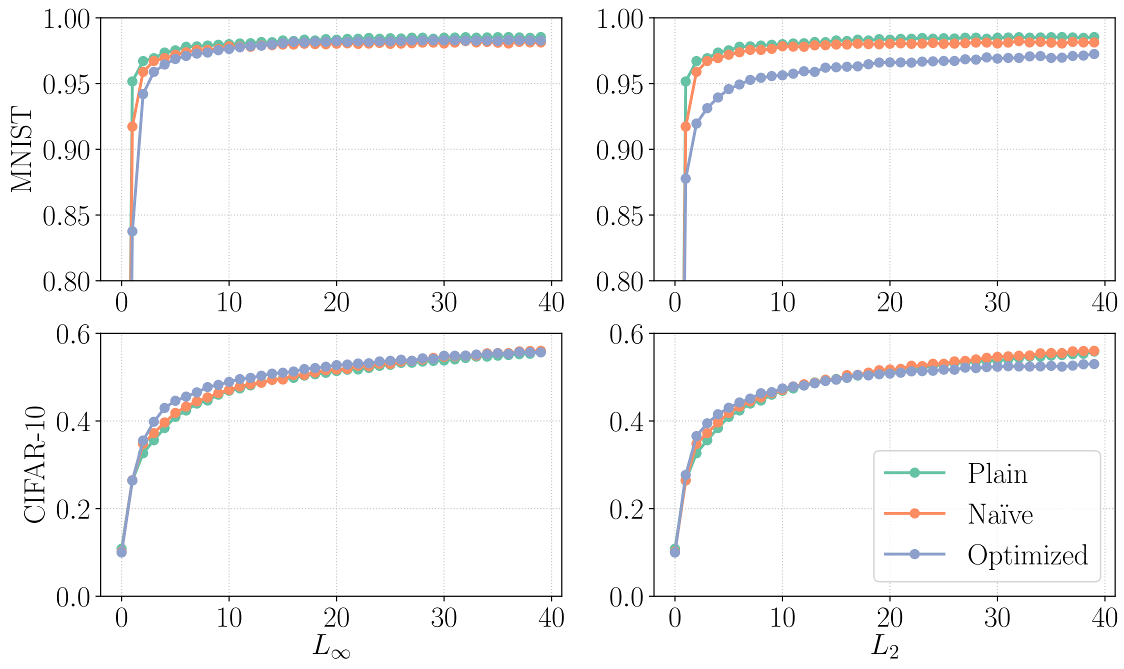


Figure 7.6: Accuracies of the end-to-end evaluation for both tasks and norm bounds.

verification. The optimistic starting improvement, where the server optimistically continues with the next round and verifies the proofs of the last round while the clients are training, mitigates this partly, but not completely. This is because there still exists some idle time where the server has to wait for the verification of the previous round to complete before continuing with the next round.

However, we can easily resolve this bottleneck by parallelizing these computations at the server using horizontal scaling on a cluster. Using optimistic starting, we only need to ensure that the time for proof verification takes at most the time that the clients take to train and generate the proofs in a round because these can then exactly be executed in parallel.

8 Conclusion

The increasing occurrence and corresponding unrest of data abuse by companies and governments show that there is a pressing need for privacy-preserving solutions that can resolve the tension between data utility and user privacy. Federated Learning is an emerging machine learning approach that attempts to solve this issue by providing the means to train machine learning models while keeping data decentralized at the clients, sharing only small, ephemeral updates specifically crafted for training the model. However, this new learning paradigm introduces a new set of privacy and integrity challenges due to its open and decentralized nature, which we must examine carefully and adequately resolve. As we rely more and more on decisions driven by machine learning algorithms in our lives, it is of the utmost importance that these algorithms are fair and trusted. Therefore, before deploying federated learning in high-stakes, privacy-sensitive applications, we must first deeply understand its robustness in the presence of adversaries.

This thesis examines the integrity of federated learning and provides a thorough analysis of the impact of integrity attacks on federated learning. We hope that this analysis can guide future solutions. This thesis also presents a new protocol that mitigates the impact of these integrity attacks. Based on insights from our analysis study, we observe that norm bounding is an effective measure to protect against these attacks. Norm bounding proves to be an effective defense against many existing model poisoning attacks under a reasonable federated learning setting. With this in mind, we design a protocol using a homomorphic commitment scheme and zero-knowledge proofs to enforce this norm bound on client updates without compromising privacy, making it compatible with secure aggregation as opposed to existing defenses. We introduce several optimizations to make the protocol practical at scale.

Our evaluation shows that the overhead of the protocol grows linearly in the number of parameters. With our optimizations, we bring the overhead of a baseline implementation of the protocol from 802 seconds down to 120 seconds per round (6.82x) for a model 62006 parameters at only a small reduction in accuracy. Hence, this shows that it is feasible to apply the protocol for training deep-learning models in practice.

8.1 Future work

The work in this thesis sheds light on some challenges that we believe are interesting to pursue in further research. This section briefly highlights some of these challenges and discusses some ideas for further research in secure and robust federated learning.

- **Theoretical guarantees.** In this work, we empirically showed that norm bounding is an effective defense against many existing model poisoning attacks. However, the work is based on empirical analysis, and we do not have theoretical guarantees that norm bounding will protect against newly discovered attacks. To get stronger security guarantees, we must complement empirical defenses with ones that provide theoretical guarantees. Defenses with theoretical guarantees provide an upper bound on adversarial success, even when better attacks are found. A similar development was seen in the field of adversarial machine learning, where empirical defenses were proposed first, followed by defenses that provide a theoretical limit on adversary success. However, the formulation of theoretical defenses in federated learning is more challenging than in adversarial machine learning, because the threat model is stronger; the adversary has white-box access to the model during training.

- **Cryptographic constructions.** In Section 5.6, we established two directions for improving the performance of the protocol and took one direction by optimizing the protocol using machine learning techniques and compression. However, we suspect even more performance gain can be achieved by using cryptographic constructions that are tailored and optimized for our setting in which a large set of untrusted parties perform secure aggregation and zero-knowledge protocols on large vectors.

- **Advanced attacks.** In our analysis, we showed that norm bounding protects against all existing integrity attacks. It is however very likely that other attacks exist, and it is useful to discover these to get an improved understanding of the integrity of federated learning models. With these insights, we can then develop additional defenses to make federated learning more robust.

Bibliography

[1] cargo bench - the cargo book. `https://doc.rust-lang.org/cargo/commands/cargo-bench.html`. Accessed: 2020-6-20.

[2] Home – waymo. `https://waymo.com/`. Accessed: 2020-5-26.

[3] Leonhard - ScientificComputing. `https://scicomp.ethz.ch/wiki/Leonhard`. Accessed: 2020-6-1.

[4] SRP: What is it? `http://srp.stanford.edu/whatisit.html`. Accessed: 2020-6-12.

[5] Tensorboard. `https://github.com/tensorflow/tensorboard`. Accessed: 2020-5-31.

[6] The hypergeometric distribution. In J. A. Rice, editor, *Mathematical Statistics and Data Analysis*, volume Third edition, page 42. Duxbury Press, 2007.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[8] A. Abdulkader, A. Lakshmiratan, and J. Zhang. Introducing Deep-Text: Facebook's text understanding engine . Facebook Engineering, Online: `https://engineering.fb.com/core-data/introducing-deeptext-facebook-s-text-understanding-engine`, 6 2016.

[9] G. Ács and C. Castelluccia. I have a DREAM! (DiffeRentially private smart metering). In *Information Hiding*, pages 118–132. Springer Berlin Heidelberg, 2011.

[10] D. Alistarh, Z. Allen-Zhu, and J. Li. Byzantine stochastic gradient descent. Mar. 2018.

[11] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. QSGD: Communication-Efficient SGD via gradient quantization and encoding. Oct. 2016.

[12] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How to backdoor federated learning, 2018.

[13] G. Baruch, M. Baruch, and Y. Goldberg. A little is enough: Circumventing defenses for distributed learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8632–8642. Curran Associates, Inc., 2019.

[14] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 1–10, New York, NY, USA, Jan. 1988. Association for Computing Machinery.

[15] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive*, 2018:46, 2018.

[16] E. Ben-sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. pages 459–474, May 2014.

[17] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct Non-Interactive zero knowledge for a von neumann architecture. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 781–796. USENIX Association, 2014.

[18] J. Benaloh. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, EVT'06, page 5, USA, Aug. 2006. USENIX Association.

[19] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo. Analyzing federated learning through an adversarial lens. Nov. 2018.

[20] A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers. Protection against reconstruction and its applications in private federated learning. Dec. 2018.

[21] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 119–129. Curran Associates, Inc., 2017.

[22] J.-M. Bohli, J. Müller-Quade, and S. Röhrich. Bingo voting: Secure and Coercion-Free voting using a trusted random number generator. In *E-Voting and Identity*, pages 111–124. Springer Berlin Heidelberg, 2007.

[23] 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 *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1175–1191, New York, NY, USA, 2017. ACM.

[24] D. Boneh and V. Shoup. A graduate course in applied cryptography.

[25] Bronshtein, I.N., Semendyayev, K.A., Musiol, G., Mühlig, H. *Handbook of Mathematics.* 2004.

[26] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, May 2018.

[27] S. Caldas, J. Konečny, H. Brendan McMahan, and A. Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. Dec. 2018.

[28] S. Caldas, P. Wu, T. Li, J. Konečný, H. Brendan McMahan, V. Smith, and A. Talwalkar. LEAF: A benchmark for federated settings. Dec. 2018.

[29] J. Camenisch, R. Chaabouni, and A. Shelat. Efficient protocols for set membership and range proofs, 2008.

[30] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous SGD. Apr. 2016.

[31] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos. DRACO: Byzantine-resilient distributed training via redundant gradients. Mar. 2018.

[32] F. Chollet et al. Keras. `https://keras.io`, 2015.

[33] N. Confessore. Cambridge Analytica and Facebook: The Scandal and the Fallout So Far. The New York Times, Online: `https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html`, April 2018.

[34] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics.

[35] G. Damaskinos, E. M. El Mhamdi, R. Guerraoui, A. Guirguis, and S. Rouault. AGGREGATHOR: Byzantine machine learning via robust gradient aggregation.

[36] H. de Valence. Merlin.

[37] O. A. H. de Valence, Cathie Yun. bulletproofs.

[38] S. Dickens. Introducing New Machine Learning Techniques to Help Stop Scams. Facebook Notes, Online: `https://www.facebook.com/notes/facebook-security/introducing-new-machine-learning-techniques-to-help-stop-scams/10155213964780766`, 3 2018.

[39] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, pages 265–284, Berlin, Heidelberg, 2006. Springer-Verlag.

[40] E. M. El Mhamdi, R. Guerraoui, and S. Rouault. The hidden vulnerability of distributed learning in byzantium. Feb. 2018.

[41] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer Berlin Heidelberg, 1985.

[42] S. Englehardt. Next steps in privacy-preserving telemetry with prio – mozilla security blog. `https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/`. Accessed: 2020-5-20.

[43] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *J. Cryptology*, 1(2):77–94, June 1988.

[44] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO' 86*, pages 186–194. Springer Berlin Heidelberg, 1987.

[45] J. Finkle and D. Seetharaman. Cyber Thieves Took Data On 145 Million eBay Customers By Hacking 3 Corporate Employees . Business Insider, Online: `https://www.businessinsider.com/cyber-thieves-took-data-on-145-million-ebay-customers-by-hacking-3-corporate-employees-2014-5`, May 2014.

[46] C. Fisher. 32 million patient records were breached in the first half of 2019. Engadget, Online: `https://www.engadget.com/2019/07/31/32-million-patient-records-breached-2019`, July 2019.

[47] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1322–1333, New York, NY, USA, Oct. 2015. Association for Computing Machinery.

[48] C. Fung, C. J. M. Yoon, and I. Beschastnikh. Mitigating sybils in federated learning poisoning, 2018.

[49] F. D. Garcia and B. Jacobs. Privacy-Friendly Energy-Metering via homomorphic encryption. In *Security and Trust Management*, pages 226–238. Springer Berlin Heidelberg, 2011.

[50] R. C. Geyer, T. Klein, and M. Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[51] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, STOC '85, pages 291–304, New York, NY, USA, Dec. 1985. Association for Computing Machinery.

[52] S. Goryczka and L. Xiong. A comprehensive comparison of multiparty secure additions with differential privacy. *IEEE Trans. Dependable Secure Comput.*, 2015.

[53] J. Groth. Non-interactive Zero-Knowledge arguments for voting. In *Applied Cryptography and Network Security*, pages 467–482. Springer Berlin Heidelberg, 2005.

[54] R. Hackett. LinkedIn Lost 167 Million Account Credentials in Data Breach. Fortune, Online: `https://fortune.com/2016/05/18/linkedin-data-breach-email-password`, May 2016.

[55] R. Haenni and O. Spycher. Secure internet voting on limited devices with anonymized DSA public keys. In *Proceedings of the 2011 conference on Electronic voting technology/workshop on trustworthy elections*, EVT/WOTE'11, page 8, USA, Aug. 2011. USENIX Association.

[56] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. Oct. 2015.

[57] A. Hard, C. M. Kiddon, D. Ramage, F. Beaufays, H. Eichner, K. Rao, R. Mathews, and S. Augenstein. Federated learning for mobile keyboard prediction, 2018.

[58] F. Hartmann. Federated learning. Master's thesis, Freien Universitat Berlin,, Aug. 2018.

[59] J. Hayes, L. Melis, G. Danezis, and E. De Cristofaro. LOGAN: Membership inference attacks against generative models. May 2017.

[60] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. Mar. 2015.

[61] M. Hirt. Cryptographic protocols - MPC part 2, 2020.

[62] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[63] M. K. Ibrahem. Robust electronic voting system using homomorphic encryption protocol and Zero-Knowledge proof. Jan. 2016.

[64] H. d. V. Isis Agora Lovecruft. curve25519-dalek.

[65] P. Kairouz, H. Brendan McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao. Advances and open problems in federated learning. Dec. 2019.

[66] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. Dec. 2014.

[67] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology — CRYPTO '96*, pages 104–113. Springer Berlin Heidelberg, 1996.

[68] J. Konečný, H. Brendan McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. Oct. 2016.

[69] J. Konečný and P. Richtárik. Randomized distributed mean estimation: Accuracy vs. communication. *Frontiers in Applied Mathematics and Statistics*, 4:62, 2018.

[70] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[71] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-Friendly aggregation for the Smart-Grid. In *Privacy Enhancing Technologies*, pages 175–191. Springer Berlin Heidelberg, 2011.

[72] Q. V. Le, T. Sarlos, and A. J. Smola. Fastfood: Approximate kernel expansions in loglinear time. Aug. 2014.

[73] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[74] M. Lei. Privacy-preserving federated learning with range checks. Master's thesis, ETH Zurich, 2019.

[75] C. Li, H. Farkhoor, R. Liu, and J. Yosinski. Measuring the intrinsic dimension of objective landscapes. Apr. 2018.

[76] P. Li, T. J. Hastie, and K. W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 287–296, New York, NY, USA, Aug. 2006. Association for Computing Machinery.

[77] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang. On the convergence of FedAvg on Non-IID data. Sept. 2019.

[78] Y. Long, V. Bindschaedler, L. Wang, D. Bu, X. Wang, H. Tang, C. A. Gunter, and K. Chen. Understanding membership inferences on Well-Generalized learning models. Feb. 2018.

[79] U. M. Martin Hirt. MPC protocols I.

[80] U. Maurer. Unifying Zero-Knowledge proofs of knowledge. *Plan. Perspect.*, 272:286, 2009.

[81] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas. Communication-Efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial In- telligence and Statistics (AISTATS) 2017*, 17 2017.

[82] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. May 2018.

[83] L. Muñoz-González, K. T. Co, and E. C. Lupu. Byzantine-Robust federated machine learning through adaptive model averaging. Sept. 2019.

[84] E. Nadilinski. Demystifying zero knowledge proofs. Devcon4, Nov. 2018.

[85] 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 *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753. IEEE, May 2019.

[86] U. D. of Health & Human Services. Sharing and utilizing health datafor ai applications. Roundtable report, U.S. Department of Health & Human Services, 200 Independence Avenue, S.W. Washington, D.C. 20201, 2019.

[87] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Berkay Celik, and A. Swami. Practical Black-Box attacks against machine learning. Feb. 2016.

[88] T. P. Pedersen. Non-Interactive and Information-Theoretic secure verifiable secret sharing. In *Advances in Cryptology — CRYPTO '91*, pages 129–140. Springer Berlin Heidelberg, 1992.

[89] K. Pillutla, S. M. Kakade, and Z. Harchaoui. Robust aggregation for federated learning. 2019.

[90] J. R. Intel® AVX-512 instructions. `https://software.intel.com/content/www/us/en/develop/articles/intel-avx-512-instructions.html`. Accessed: 2020-6-15.

[91] G. Rauch. Socket.IO.

[92] S. Ravi. Custom On-Device ML models with Learn2Compress. `https://ai.googleblog.com/2018/05/custom-on-device-ml-models.html`. Accessed: 2020-5-18.

[93] L. Reyzin, A. D. Smith, and S. Yakoubov. Turning HATE into LOVE: Homomorphic ad hoc threshold encryption for scalable MPC. *IACR Cryptology ePrint Archive*, 2018:997, 2018.

[94] A. Rial and G. Danezis. Privacy-preserving smart metering. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, WPES '11, pages 49–60, New York, NY, USA, Oct. 2011. Association for Computing Machinery.

[95] E. Roth, D. Noble, B. H. Falk, and A. Haeberlen. Honeycrisp: Large-scale differentially private aggregation without a trusted core. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, pages 196–210, New York, NY, USA, 2019. ACM.

[96] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, May 2014.

[97] C. P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, Jan. 1991.

[98] D. Shanks. Class number, a theory of factorization and genera. In *Proceedings of Symposium of Pure Mathematics*, 20, pages 415–440, 1969.

[99] S. Shen, S. Tople, and P. Saxena. Auror : defending against poisoning attacks in collaborative deep learning systems. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 508–519. ACM, Dec. 2016.

[100] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. Oct. 2016.

[101] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(56):1929–1958, 2014.

[102] Z. Sun, P. Kairouz, A. T. Suresh, and H. Brendan McMahan. Can you really backdoor federated learning? Nov. 2019.

[103] O. Thakkar, G. Andrew, and H. Brendan McMahan. Differentially private learning with adaptive clipping. May 2019.

[104] R. Tomsett, K. Chan, and S. Chakraborty. Model poisoning attacks against distributed machine learning systems. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 110061D. International Society for Optics and Photonics, May 2019.

[105] S. Truex, L. Liu, M. E. Gursoy, L. Yu, and W. Wei. Towards demystifying membership inference attacks. June 2018.

[106] K. Wiggers. Uber details VerCD, the AI tech powering its self-driving cars. `https://venturebeat.com/2020/03/04/uber-details-vercd-the-ai-tech-powering-its-self-driving-cars/`, Mar. 2020. Accessed: 2020-5-26.

[107] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[108] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett. Byzantine-Robust distributed learning: Towards optimal statistical rates. Mar. 2018.

A Mathematical Background

Hypergeometric distribution

Given a set of N elements where K elements have a specified feature that we would like to draw, the hypergeometric distribution describes the probability of having k successes in n draws from this set, without replacement after drawing an element.

Definition A.1. A random variable X follows the hypergeometric distribution if its probability mass function is given by,

$$\Pr(X = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}$$

where N is the population size, K is the number of success states in the population, n is the number of draws and k is the number of observed successes [6].

Negligible function

Intuitively, a negligible function is a function whose output goes to zero as $x \to \infty$, but does so faster than the inverse of any polynomial [24].

Definition A.2. A function $f : \mathbb{N} \mapsto \mathbb{R}$ is called *negligible* if for all $c \in \mathbb{R}_{>0}$ there exists $n_0 \in \mathbb{Z}_{\leq 1}$ such that for all integers $x \leq n_0$, we have:

$$|f(x)| < \frac{1}{x^c}$$

Cryptographic hash function

Definition A.3. A function $f : \mathcal{I} \mapsto \mathcal{O}$ is a cryptographic hash function if it maps inputs from an arbitrary length to outputs with a fixed size, i.e. $\mathcal{I} = \{0,1\}^*$ and $\mathcal{O} = \{0,1\}^{128}$, and f is *one-way*. That is, given an output o, it is infeasible to find a corresponding pre-image $i \in \mathcal{I}$ such that $f(i) = o$.

MPC Security

Definition A.4. Given a multi-party computation protocol and a specification that defines the behavior with regards to a trusted third party, a multi-party computation protocol is secure if the adversary cannot achieve anything in the protocol that they could not achieve in the specification [79].

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Scalable and Robust Privacy-Preserving Federated Learning

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Lycklama à Nijeholt	Hidde

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
23/06/2020 Zürich	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.