Federated learning cryptography

Roberto García Alvira

Master thesis
Spring 2025

Supervisors:
Prof. Ion Petre, Docent Yury Nikhulin

DEPARTMENT OF MATHEMATICS AND STATISTICS

In accordance with the University of Turku's quality system, the originality of this publication has been verified by Turnitin Originality Check system.

UNIVERSITY OF TURKU
Department of Mathematics and Statistics

Roberto García Alvira: Federated learning cryptography
Master thesis
Mathematics
Spring 2025

SUMMARY

The main topic of this thesis is the review and analysis of federated learning, a framework for collaborative machine learning and deep learning model training. The dissertation will also link federated learning with zero knowledge cryptography, exploring how this type of cryptography can enhance the security of federated learning protocols.

The thesis will start with a historical overview to situate motivation of the development of federated learning in the actual research scene, which will be followed by a general introduction of federated learning, categorizing its different courses of action. Next, there will be a review on the privacy and security of these kinds of frameworks, exploring their main risks and threats, while also analysing what kind of security mechanisms can be implemented to combat these. Afterwards, the thesis will switch its focus to the mathematical bases of zero knowledge algorithms, specifically explaining methods based in the discrete logarithm problem over elliptic curves. These mathematical principles will help set up the grounds for the analysis of the zero knowledge algorithm Groth16 [52], based on a protocol named ZK-SNARK. Finally, the thesis will culminate with the study of a specific novel protocol named ZKPoT [42], which combines blockchain, federated learning and ZK-SNARKs. Specifically, ZKPoT uses the Groth16 based ZK-SNARK as a subprotocol in the algorithm. The thesis will finish with an evaluation of how zero knowledge cryptography can enhance federated learning frameworks, fixing some of their flaws and also will explore current and future application of this type of technology in the real world.

Contents

1 Federated Learning

1.1 Historical Overview

The first important discovery on the path of ML creation was in 1943, when Warren McCulloch, a neuroscientist and Walter Pitts, a logician, published the stuy [86]. This paper named *"A Logical Calculus of the Ideas Immanent in Nervous Activity"* was inspired by the functioning of biological neurons, in order to create a mathematical model that would imitate neural activity. The mathematical interpretation behind this idea was that neurons could be represented as binary threshold units, either firing (on) or not firing (off), based on a weighted sum of inputs. With this simple idea, this paper set the theoretical basis for the creation of artificial neural networks, as they explored how complicated computations could be performed in an easier way using networks of simple units. Essentially, they proved that even simple neural structures could compute any function that a Turing machine could, creating a link between biological and computational models of thought. Despite the lack of technological material at that time, their work would inspire future researchers to study learning mechanisms and how to implement them in machines.

Following the trend of neuronal activity inspiration, Donald Hebb published in 1949 his study [112], where he introduced new theories on neuron interaction. This paper in neuroscience explains how neural connections are reinforced when they are being activated repeatedly. This principle of neural activation became fundamental later on when understanding the development of learning algorithms in artificial neural networks.

After many years of research in the field, in the year 1950 Alan Turing published a compilation of his work in one of the most groundbreaking papers the computing field has ever seen. This famous paper called *"Computing Machinery and Intelligence"* [122] elaborated on the relevant term Turing Machine and proposed the Turing Test as a way to define and measure machine intelligence [63]. Turing argued that if a machine and a human could have a conversation in such a way that the human could not distinguish between human and machine responses, the machine could be considered *"intelligent"*. Therefore, this test was a suggestion on how to formalize and define whether or not the responses given by a human were part of the computable space. Turing's participation in the field of computing motivated from his work during World War II, where he helped break the German Enigma code using early computers. This experience during the war convinced Turing that machines could be used and programmed to solve complex problems, emulating human behaviour. Therefore, in his paper he proposed the term *"learning machines"*, which provided strong theoretical and philosophical foundations for artificial intelligence in general and machine learning. Also, the Turing test served as a framework for future research in AI, in the case of creating machines that would be able to reason and communicate like human beings, setting up future advances in natural language processing and machine learning.

The first time that the term artificial intelligence was officially used dates back to the summer of 1956 in the Dartmouth Conference. John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon organized the event at Dartmouth College in New Hampshire. In this conference they had a proposal to the community [85], in

which they stated that *"every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it"*. The main goal of the conference was to gather top researchers in mathematics, cognitive science, and computer science in order to investigate further the possibility of creating machines that will be able to simulate human behaviour. The organizers were optimistic that significant research advance could be made in a decade, however this was not the case. Despite this the conference atracted funding from goverment and military agencies and also introduced important ideas that would prove useful in future AI research, such as symbolic reasoning and machine learning.

Later in that same decade the computer scientist Arthur Samuel developed a program that learned how to play checkers [108], marking the first occasion where a machine was proven to have self-learning skills. This demonstrated that computers could learn and enhance their performance over time without being reprogrammed, therefore showing early signs that machine learning was definitely possible to achieve.

Another important breakthrough at that time was the Perceptron algorithm [106], created by the psychologist Frank Rosenblatt in 1957. This algorithm was an early neural network model, consisting of a single layer of artificial neurons connected by adjustable weights, which was used for recognizing patterns and learning from them. Perceptron could learn to classify data into two categories by adjusting these weights based on feedback from incorrect predictions, which is now known as supervised learning. The main inspiration for this model was the ability of the human brain to process visual information and act accordingly to those stimuli. The perceptron influenced very positively the neural network scene as it meant a significant step forward towards the development of algorithms that could learn from data.

Additionally, the first statistical methods that later would be the backbone of machine learning were already developed during the 1950's and 1960's. For example, Bayesian methods are key in probabilistic inference and they were introduced for this matter in Machine Learning in the year 1964 by R.J. Solomonoff, with his paper [115]. The core of Bayesian statistics is the Bayes' Theorem, which describes the probability of a hypothesis H happening based on observed evidence E:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)},$$

where $P(H)$ is the prior probability of the hypothesis, $P(E|H)$ is the likelihood or probability of observing evidence given the hypothesis, $P(E)$ is the probability of the evidence happening under all possible hypotheses and $P(H|E)$ is the posterior probability, ie, the probability of the hypothesis after observing the evidence. Bayesian statistics are used in many circumstances in AI, such as for inference, creating bayesian networks or optimizing techniques. This type of statistics is used so much in AI and ML because it naturally provides a measure of incorporating prior knowledge into a system, which can provide a measure of probability for a prediction.

In 1966, Joseph Weizenbaum created ELIZA [130] an AI program that was capable of processing natural language. ELIZA simulated human conversations by analysing input for keywords and generating already scripted responses. The most

famous version of ELIZA was when it simulated a Rogerian psychotherapist, being able to reply to open questions like, *"How does that make you feel?"*. However, ELIZA was not truly intelligent as it had no understanding of the language and the meaning of the words, but this AI proved that simple pattern matching techniques could create the illusion of intelligence in a machine. Weizenbaum commented that users quickly attributed human behaviour qualities like understanding and empathy to ELIZA, despite the machine being programmed just with premade responses. This raised some ethical considerations on how human and machines could interact between each other in the future [64].

Machine learning research had a drought period during the 1970's named *"AI winter"*. This period saw reduced funding and interest in machine learning research because of many reasons. First of all, in 1969 Marvin Minsky and Seymour Papert published the book *"Perceptrons"* [92], where they mathematically demonstrated that the single-layer perceptrons, the ones introduced by Rosenblatt, could not be used to solve simple problems, like computing the XOR function. Therefore this discouraged further research into neural networks. Also, researchers switched their focus away from neural networks into symbolic AI. Symbolic AI represents knowledge using symbols and explicit rules in order to make decisions, therefore is easier to understand and implement but it did not translate well when applied to the real world, frustrating the community.

Later, in 1973 the British government published the Lighthill Report [1], where they stated the situation of AI research and criticised its poor progress. Their main point of argument was that AI failed to produce practical solutions outside of specific tasks. Despite all the theoretical development, AI was not able to handle real world complexity.

This report unchained significant cuts in funding from the UK government and influenced other countries to do the same, setting back AI research and improvement. This report was one of the bigger contributors to the *"AI winter"* period.

Additionally, at this time there was a lack of computational power that machine learning models needed to perform correctly, and also researchers did not have big enough datasets in order to effectively train these algorithms. Therefore, without these resources it was complicated to demonstrate if machine learning was truly powerful and useful in the future.

The 1980's marked the end of the *"AI winter"* drought period and interest was regained on research. One of the major reasons was the commercial success of expert systems, which were rule based programs designed for specific domains, that tried to simulate human decision making to solve different tasks in this domain. Expert systems had a different approach about AI than the previous inventions. Instead of simulating general intelligence, expert systems focused of specific tasks that required specialized knowledge, such as making medical diagnosis, geological analysis, and financial forecasting.

An example of expert system is XCON, the first one that was commercialized by Digital Equipment Corporation (DEC). XCON was an expert configurer that would configure computer systems based on customer orders. It was based on hundred of if-then rules that automated this task that previously required a trained human to perform, reducing human error and saving money with the automation process.

However, expert systems had also their shortcomings, as they were not flexible in task solving, required constant maintenance to update rules based on need and also did not have the ability to learn by themselves utilising new user data.

Despite their limitations, the expert system market was estimated on 1 billion dollars by 1986's, which inevitably rose awareness about the world of AI and marked the start of an AI boom in the 1980's.

Some years after, in 1986 David Rumelhart, Geoffrey Hinton, and Ronald Williams published a paper about the backpropagation algorithm [80]. This was a major breakthrough in the AI field and this revived the interest of the community about neural network research. This algorithm enables multi-layer neural networks to update their internal weights based on the errors observed at the output layer. These errors are then propagated backwards in the layers of the neural network and each connection weight is adjusted using gradient descent.

In years prior, neural networks were only limited to structures with a single layer, like the Perceptron, so the backpropagation algorithm was able to provide a solution for the creation of more complex neural networks. These new multi-layer networks could solve more complicated and non-linear problems, making them able to solve tasks like recognizing speech patterns and handwritten characters. Hinton et al even probed that the backpropagation algorithm made neural networks more effective than expert systems when solving certain tasks. Backpropagation was a successful algorithms and it marked the beginning of the transition from symbolic AI to learning-based AI, laying the foundation for modern deep learning.

Despite all the progress in the field during the decade, the AI community entered a second *"AI winter"* in the late 1980's. A big reason of this recession was the downfall of the expert systems, that as stated, became too hard for companies to maintain due to high cost and complexity. Also, some other hardware and specific machines used for AI, like Lisp machines became obsolete, as normal computers soon became more affordable and powerful. Therefore, funding for AI and research again declined, and many AI companies and startups abandoned the business due to this commercial crisis. In the academic research picture, the interest also reduced as machine learning was computationally too expensive resource wise.

In the decade of the 1990's, switched to a statistical and data-driven approach. One of the catalysers of this switch was the application of hidden markov models (HMMs) to speech recognition [44]. HMMs are probabilistic methods that model how systems evolve over time, therefore researchers used them to represent spoken language or sequences of data, and created systems that could transcribe human speech with better accuracy than other rule based systems. The key takeaway for this advancement was that researchers realized that language could be modelled probabilistically, making machines learn language patterns from datasets instead of basing their operating procedures in rules. This marked the shift from symbolic AI, which was rule based, to data driven machine learning. This breakthrough improved performance in language processing, computer vision, and served as basis for future advances in sequence modelling, including recurrent neural networks.

Another influential breakthrough introduced in the middle of the 1990's were the support vector machines (SVMs), created by Vladimir Vapnik and Corinna Cortes [25]. SVMs are supervised machine learning models that use geometrical techniques

to identify the hyperplane that separates the data into different classes the best. They introduced the *"kernel trick"*, which mapped data that was non-linearly separable into higher dimensional spaces to it could be classified there. This technique made SVMs really effective for high complexity classification problems, such as handwriting recognition and image classification. SVMs quickly were established as a standard tool in AI and stayed popular well into the 2000's.

In 1997 a landmark moment in AI history happened, when Deep Blue, an AI developed by IBM, defeated world chess champion Garry Kasparov in a six game match [3]. This AI was not based on learning algorithms, but rather on brute force. Deep Blue could evaluate up to 200 million chess positions per second, using evaluation functions handcrafted by the IBM researchers and a huge database of chess openings and endgames.

Ethically, Deep Blue supposed a change on how machines are perceived by humans, even Kasparov quoted: *"For the first time in the history of mankind, I saw something similar to an artificial intellect"*. Even though Deep Blue relied on computational power instead of true intelligence, it demonstrated that machines could actually outperform humans in complex tasks, therefore inspiring further exploration in AIs that could play complex games and also complex decision making models.

In the decade of the 2000's big data and machine learning saw a big rise, specially when in 2006, Geoffrey Hinton introduced the concept of deep belief networks (DBNs) [61]. This type of network is trained using unsupervised learning, and used an efficient method at the time called *"contrastive divergence"*, that allowed DBNs to train deep networks layer by layer. This breakthrough was key for solving the problem of training deep networks, as previous attempts did not success due to the vanishing gradient problem. This approach allowed those deeper networks to be trained, converge and learn complex patterns in data. With increased computing power and larger datasets available, summed up with the advances in research, deep learning had a resurgence and the switch to data driven AI was noticeable.

As stated AI was shifting to a data-driven approach, therefore large amounts of data were needed to train AI effectively. That is why Fei-Fei Li and her team initiated the ImageNet [30] project, to create one of the most thorough and complete databases of images. The ImageNet database provides a massive amount of labelled data used for developing object recognition software, as it contains millions of labelled images across thousands of categories. The immense scale and great quality of this dataset facilitated research in computer vision, especially when training deep learning models to recognize and classify the objects contained in images [64].

Particularly, in 2012 Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton created a deep convolutional neural network named AlexNet [72] and won the ImageNet competition of image recognition, with a 10% improvement compared to previous algorithms. AlexNet is an eight layer neural network, which introduced some innovations like rectified linear unit (ReLU) activation functions, dropout for regularization, and the use of GPUs to accelerate training. AlexNet was this successful because of the existence of large datasets, like ImageNet, and also because it used GPUs for training which increased significantly the computational performance and efficiency. This proved the effectiveness of deep convolutional networks and GPU acceleration, which triggered the rise of deep learning. After this, AI quickly started

to surpass human performance in computer vision, language translation and other fields.

Another framework for training deep generative models was introduced in 2014 by Ian Goodfellow et al. The framework was called Generative Adversarial Networks (GANs) [51] and it consisted of two neural networks, one called the generator and the other the discriminator, that are trained through adversarial training both at the same time. The function of the generator neural network is to create synthetic data, that then, the discriminator evaluates and decides if the data is real or fake. Therefore, these two networks compete against each other improving their performance together over time. GANs were created with the purpose of finding models that could generate high-quality and realistic data by themselves from scratch, in the form of images, music or text. GANs soon became then a powerful tool for AI for creating hyper-realistic images, videos and artwork, therefore marking the change to creative AI, where machine did not just recognize patterns but also were able to generate content by themselves.

The following year, in 2015, Kaiming He et al introduced the Residual Network (ResNet) [59] while working at Microsoft Research. This algorithm was taken to the ImageNet competition and won it with an outstanding error rate of just 3.6%, which surpassed human level performance for image recognition. ResNet addressed efficiency problems that previous AIs had, like the vanishing gradients problem, by introducing the skip connections procedure. This procedure allowed gradients to get transmitted more effectively during the backpropagation part of the algorithm. ResNet then proved that well structured deep networks could outperform shallower ones. Therefore, this fuelled research into deep learning, following the trend of the decade of the 2010's.

The next biggest achievement for game modelling AIs after Deep Blue defeated Kasparov in 1997 was, when in 2016, the AI named AlphaGo developed by DeepMind defeated the world champion Go player Lee Sedol in a five-game match [14]. Go is a 3000 year old board game original from China, with the majority of its community comes from east Asia. It is a complex board game, also for computers to model, due to its huge search space, as there is more board positions than atoms in the universe [120], and due to its need for intuitive play.

Contrary to previous game engines, like Deep Blue, who relied on brute force search of all the possible positions, AlphaGo learned patterns and strategies from millions of games where it played itself. AlphaGo used a combination of reinforcement learning and Monte Carlo tree search (MCTS) in order to evaluate positions and select moves. AlphaGo's victory was a milestone for AI, as it showed the ability of AI to handle complex and dynamic environments, without relying on brute force, making it closer to human thinking. It also highlighted the potential of reinforcement and deep learning for real-world applications, as game theory is often used to model real life scenarios.

A year later, in 2017, Vaswani et al. introduced the transformer model in their paper named *"Attention is all you need"* [123]. The transformer framework replaced recurrent and convolutional layers with self-attention mechanism. This allowed the model to process entire sequences of data all at the same time, making leaning more efficient when studying long range dependencies in languages and texts.

The self-attention mechanisms previously mentioned were the key innovation of this framework, which allowed the model to give different weights to the importance of words in a sentence. Therefore, transformers became the staple architecture for natural language processing (NLP), being the core of model like BERT [32] (Bidirectional Encoder Representations from Transformers) or GPT (Generative Pre-trained Transformer) [137]. Transformers were created to overcome the limitations of RNNs and CNNs, as these could not process well enough long-term dependencies. After being introduced, transformers revolutionized language modelling, improving AI performance in translation, text generation and question answering.

As mentioned in the previous paragraph, transformer helped build many natural processing models and one of those was the Bidirectional Encoder Representations from transformers or BERT. BERT [32] was introduced in 2018 by J. Devlin et al., working for Google and the model was designed to understand the context of words in a sentence more effectively using bidirectional language processing. The bidirectional processing meant that BERT jointly analysed text both from both left-to-right and right-to-left. BERT was trained using unsupervised learning on massive quantities of texts, including the full Wikipedia, where the model learned to predict missing words and how sentence relate with each other. Other previous models processed text sequentially, that is why the predicting and attention based structure supposed a big improvement in the context of question answering, sentiment analysis, and language translation.

Another transformer based AI was launched in 2020 by OpenAI, by the name of GPT-3 [15] (Generative Pre-trained Transformer 3). This AI contained 175 billion parameters, making it the largest language model ever created at that time. GPT-3 demonstrated outstanding skills on language generation, including the ability to write essays, generate code, answer questions and even creating poetry, making clear that could understand human feelings.

The biggest innovation under GPT-3's success was the ability to perform few-shot and zero-shot learning, meaning that the model could generate its outputs based on only a few examples or no examples at all. This type of learning showed that using diverse types of texts in large scale for the pre training could enable general language understanding and generation. Nowadays, five years later, ChatGPT is the engine that uses this AI and it has become famous worldwide for its ability to solve diverse varieties of problems with a human like intelligence. The version of GPT has evolved and now the available versions for the users are 4o and 4.5, while OpenAI keeps updating and improving ChatGPT.

In 2021, DeepMind developed AlphaFold [66], which solved the protein folding problem, one of the biggest challenges in biology throughout the last century. AlphaFold was based in a deep neural network that predicted the 3D structure of proteins just from their amino acid chain sequences, with an accuracy almost parallel to experimental. This was an incredible breakthrough in the biology and bioengineering sector and it accelerated fields like drug discover or the understanding of diseases.

Finally, one of the latest AI breakthroughs happened in 2022 when OpenAI introduced DALL-E [104]. This AI has a generative model that is capable of creating high-quality images just based on text prompts, which has proven to have

an exceptional ability to generated realistic and complex artistic images based on those prompts. DALL-E was based on a variant of the transformer architecture, which combined elements of GANs and attention mechanism. The motivation behind DALL-E's creation was to explore the boundaries of creative AI and expand the capabilities of generative models, combining language and vision.

To understand the motivation behind the development of federated learning, it is important to come back to the decade of the 2010's. In this decade despite all the progress that had been made, machine learning was still not widely adopted enough for practical use in personal devices, therefore its usage was usually restricted to people who possessed the adequate hardware and software.

This shortcoming changed when deep learning became feasible later in the 2010's due to the advances on neural network based machine learning. Suddenly, machine learning was integrated in plenty of software devices and applications utilised by the average user. Being able to work with huge amounts of data from all the users was great for the development of machine learning algorithms in terms of improving results and efficiency.

Usually, all these machine learning algorithms would work on a dataset that is located in one big server, where they run and improve on each iteration. However, with the spreading of machine learning algorithms to the average user, data privacy became a major concern with this centralized training approach for machine learning models. Users would send their data to the main server for the algorithms to be trained, however this data could contain sensitive information that the user might not want to disclose, therefore breaking huge data privacy policies. In response to this misuse of personal data, Google introduced in 2016 the term *"federated learning"* in its blog [87].

Google proposed to solve the data privacy issue with the following simple idea: instead of bringing the data to the model, they would bring the model to the data. So, the training of the machine learning model would work like this: They would select a set of users which will download the current model, they will improve the model by learning from the data in their devices. After the training is done an update of the model its encrypted and sent to the cloud, where it is immediately averaged and compared with the updates from other users to improve the shared model. All of this while maintaining the data in the user's device and without having to store any updates in the cloud.

After Google's proposition, the community started working on the idea of federated learning, as it solved the privacy issue they had been worried about. In 2017, the algorithm *FedAvg* was introduced in the paper [88], written by H. Brendan McMahan et al. This algorithm will set the basis for how federated learning would work, as it allowed an efficient model training across multiple devices, creating the foundation for federated learning frameworks.

With the massive gain of popularity and usage that AI and machine learning have reached now in the 2020s, federated learning has also become more widespread. Federated learning expanded to areas beyond mobile apps, such as healthcare or finance, where research groups work on collaborative AI projects. Now that the procedures and frameworks for federated learning were settled, the community shifted its focus to improve its main distinctive, data privacy and security. Making systems more

secure and robust made sure that common issues like model poisoning or data leakage were eradicated and enhanced user trust in this type of machine learning. The way federated learning improved its security and robustness was by implementing better and more complex cryptographic systems into its encrypting phase. Some of these cryptographic techniques were homomorphic encryption and zero-knowledge proofs, and these techniques will be expanded on in this dissertation.

Right now federated learning is a field that is gaining popularity and its in constant development. There are some areas that the community is working on to enhance federated learning in the future. There are some studies that emphasize the need for federated learning to be more personal for the user, satisfying its preferences and correlating with its data better. Right now the models are averages of the updates made from all the users, thus, this trend aims to improve personal model performance while still keeping data privacy intact. Another important trend is trying to make federated learning compatible with edge computing. Instead of sending updates to the central server and taking more time to update the whole model, working with edge computing will enhance data processing time and would bring much faster and efficient updates to the user's device. Finally, as federated learning is becoming more prevalent nowadays, the algorithms and frameworks must keep up with data protection regulations, such as GDPR. Building correct and ethical frameworks is key in the future development of federated learning.

As discussed, federated learning is an up and coming field that has seen a great rise in usage due to the growing need for privacy and security measures in the technological industry and machine learning in particular. In this dissertation, the main focus will be on data privacy and the security aspect, as the cryptographic algorithms that are behind the robust federated learning frameworks will be studied more in depth.

1.2 Structure of the FedAvg Framework

This section will explore a formal description of the functioning of the *FedAvg* algorithm, based on the *FedAvg* original paper [88] mentioned before in this dissertation and on the thesis [57], written by Florian Hartmann in 2018.

Machine learning algorithms start with an optimization problem. In the case of supervised learning, there is a set of n inputs $(x_i)_{i=1}^n$, that have their corresponding n labels or outputs, $(y_i)_{i=1}^n$. The data is sampled and has a distribution that is unknown, therefore, the goal of the algorithm is to find a function f that maps the input values and the labels as well as possible and that the function also fits correctly new data with the same distribution that will be sampled in the future. Different techniques are used to do this but for example, f could be assumed to be linear and then adjust the different coefficients for the function to fit the dataset.

Also, there exists unsupervised machine learning, where the output values are missing and the optimization for the f function is defined differently. However, in order to run the traditional machine learning algorithms, the n data points have to be stored in one unique server. Instead, as explained prior, the n data points are spread out into the appliances of the users, where model updates are made. This type of training is not totally decentralized, because the algorithm has a phase

that is run by the main server. This is denominated a *federated system*, because a federation of users carry out a major part of the model training, but a central server is still managing the system.

The federated learning system framework has n data points, divided between the devices of k clients. Each user has a distinct number of data points n_i.

The algorithm is initialized with the first iteration. The main server selects a subset of K users from the pool. A copy of the actual global model parameters $\theta \in \mathbf{R}^m$ is sent from the main server to these K clients, who then use their local information to train their own model update. The update of the i-th client is denoted $H_i \in \mathbf{R}^m$. After the training in all the personal devices, the updates are sent back to the server.

Now the server combines all the updates received in a federated average, giving the name to the algorithm. It does it by calculating the mean of all the gradients, weighted by the amount of data points the user has utilized to train their gradient:

$$\theta \longleftarrow \theta - \sum_{i=1}^{K} \frac{n_i}{N} H_i,$$

where, $N = \sum_{i=1}^{K}$ is the entire amount of data points utilised in this iteration. After the iteration has finished and the model is updated, a new iteration can begin. Figure 1 is a simplified visual representation of all the steps in the algorithm.

Some observations about the framework are: Only K users are required to make updates in every iteration. As it would be ideal for the improvement of the model to ask all the users for data and training, it is also very costly to perform, as the network could have a large number of users, possibly going over several millions. Therefore, asking just a certain amount of users enables the training to be more viable and capable of running many iterations.

The algorithm is iterated until the model parameters converge to a determined criteria agreed by the owners of the model. However, in some situations it might make sense to keep the algorithm running indefinitely in order to keep the training up to date. In case the users change in preferences, the model will automatically adapt correspondingly and this will ensure that clients have a good model to use locally in their devices.

1.3 Categorization of Federated Learning

After defining the framework of federated learning and knowing how the FedAvg algorithm works, it is possible now to define a classification for federated learning based on the characteristics and data distribution.

Let $\mathcal{D}_i$ be a matrix representation of the dataset from user i. Every row represents a sample and each column is a feature. Some datasets may also contain label data. Then, $\mathcal{X}$ is denoted as the space of features, $\mathcal{Y}$ as the space of labels and $\mathcal{I}$ as the sample ID space. As an example, $\mathcal{I}$ can vary depending on the field the data is being obtained from, in finances, the labels might be user's credit or in education they can be the degree of the student. It is needed that this space to denotes if the data is being obtained from a similar source or not. The triplet $(\mathcal{X}, \mathcal{Y}, \mathcal{I})$ constitutes the

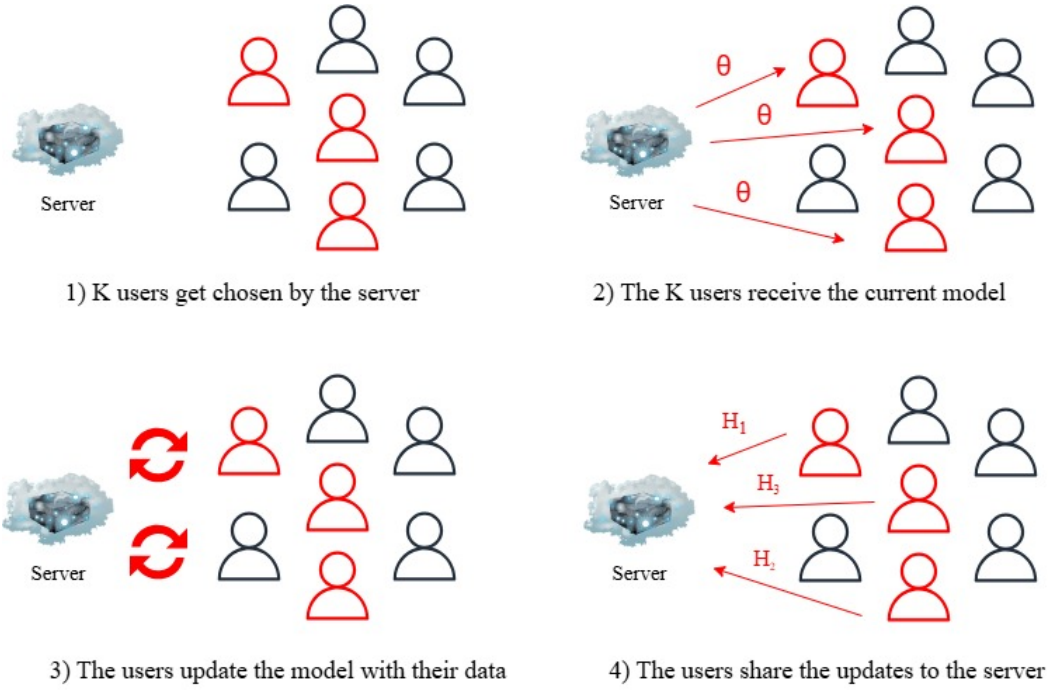


Figure 1: Communication round for the federated learning algorithm. Based on [57].

complete training dataset and based on these features, federated learning algorithms can be classified into horizontal federated learning, vertical federated learning and federated transfer learning [134].

1.3.1 Horizontal Federated Learning

Horizontal federated learning, also known as sample-based federated learning, is introduced in the scenarios in which datasets share the same feature space but different space in samples [134]. In real life, this could be the case of two orthopaedic hospitals in two different cities: they attend patients that have similar health conditions but they are in different places [62]. In 2017, Google built a horizontal federated learning framework, that helped with Android phone model updates [57, 88]. The code is open source and can be checked online in the references made by the author, Florian Hartmann [56, 107, 55]. With this framework, just a single user utilising an Android phone updates the model parameters locally in the device and then uploads the parameters to the Android cloud, thus collaboratively training the centralised model with the rest if the users. Horizontal federated learning can be formally defined as:

$$\mathcal{X}_i = \mathcal{X}_j, \quad \mathcal{Y}_i = \mathcal{Y}_j, \quad \mathcal{I}_i \neq \mathcal{I}_j, \quad \forall \mathcal{D}_i, \mathcal{D}_j, i \neq j.$$

1.3.2 Vertical Federated Learning

Vertical federated learning, also known as feature-based federated learning, is applicable to cases where two datasets share the same sample ID space but differ in the feature space [134].

Despite the challenges in real life for this type of implementation, practical examples can be found: consider two different companies one is a bank, and the other is an e-commerce company, that both operate in the same city. Their user databases are likely to contain the majority of the residents of the area, so the intersection of their user space is large. However, there is a significant difference between their feature spaces, since the bank records the user's revenue and expenditure behaviour and credit rating, and the e-commerce gathers the user's browsing and purchasing history. These two companies could collaborate and create a prediction model for product purchases based on their user and product information, implementing in this way vertical federated learning.

It is important to note that when this federated learning method is used in real life, the personal data of the users is held private, as only other user features are shared for the training of the model. This increased privacy is an interesting quality and one real life example of this vertical federated learning implementation comes from the banking sector in China, specifically WeBank [129]. This bank implemented a vertical federated learning model using the platform FATE [37] (Fed AI Technology Enabler), that utilised this technology in order to create a risk management model of credit in small and micro enterprises [38].

Finally, vertical federated learning can be formally defined as:

$$\mathcal{X}_i \neq \mathcal{X}_j, \quad \mathcal{Y}_i \neq \mathcal{Y}_j, \quad \mathcal{I}_i = \mathcal{I}_j, \quad \forall \mathcal{D}_i, \mathcal{D}_j, i \neq j.$$

1.3.3 Federated Transfer Learning

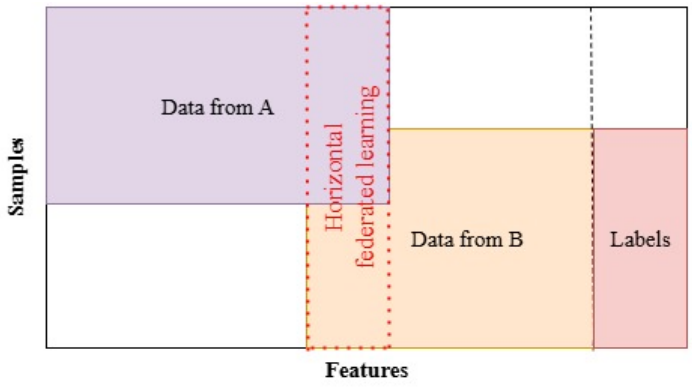
Lastly, federated transfer learning is applied to situations when the two data sets differ in their sample space as well as their feature space [134].

Think of two institutions, one is a bank located in Germany and the other one is an online commerce company from Finland. Naturally, because of geographic restraints, the user spaces of both companies have a slight overlap. Additionally, since they are distinct types of businesses, just a few of their features in the feature space overlap. This is the case where federated transfer learning can be applied and be useful for training models.

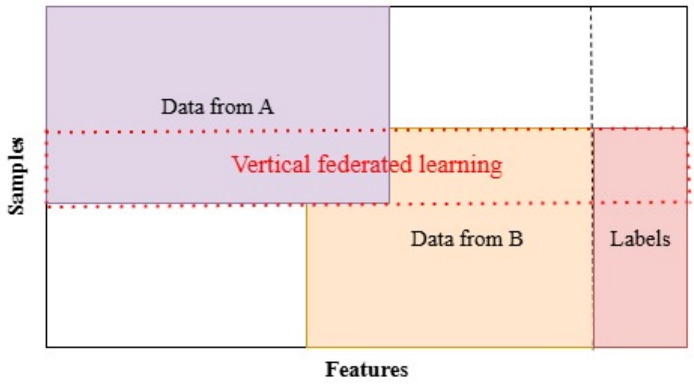
An algorithm called FedHealth [54] was developed by some Chinese researchers to enhance healthcare data protection. They used federated transfer learning to tackle the two following challenges: firstly, solve how to perform data aggregation from different sources without compromising data privacy from the users, and also, make the model be more personal and satisfy the user's needs, as models trained in the cloud usually fail in this aspect. The authors hope that this algorithm will solve data protection issues, as it has been a frequent issue that personal data patient data is accessed without permission, due to the increasing usage of IoT and its cryptography not being strong enough to combat those attackers.

Finally, federated transfer learning can be formalized with the following expression:

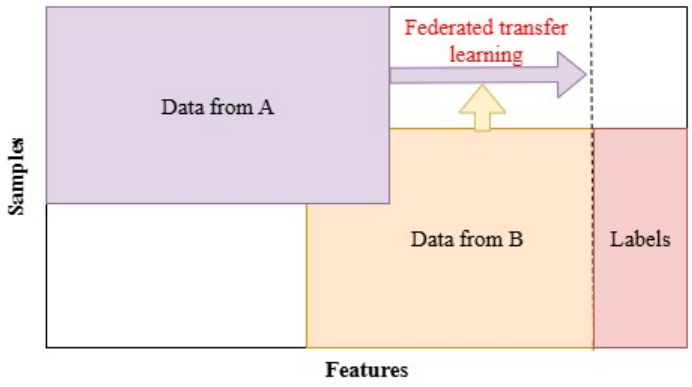
$$\mathcal{X}_i \neq \mathcal{X}_j, \quad \mathcal{Y}_i \neq \mathcal{Y}_j, \quad \mathcal{I}_i \neq \mathcal{I}_j, \quad \forall \mathcal{D}_i, \mathcal{D}_j, i \neq j.$$



a) Horizontal federated learning



b) Vertical federated learning



c) Federated transfer learning

Figure 2: Category classification of federated learning. Based on [134].

1.3.4 Other Categories

Federated learning algorithms can also be classified in other different complementary categories depending on different criteria like centralization of the data or nature of the participants [68, 46, 50].

Depending on the centralization of the data being handled used for model training, there exists:

- **Centralized federated learning:** These are algorithms that, for handling the training, use a central server. In this case, the users train their models locally in their devices with their own data and then forward the model updates to the central server. After collecting all the updates, they are aggregated by the central server, creating a global update of the learning model and next, they are sent back to the clients in order to complete the next training round. Some important features of this type of model is that it is well coordinated and the aggregation process is more simple as the central server has full control of the training process, which makes it simple to run the protocol. However, this framework is more exposed to attacks due to all the information being sent, stored and handled in the same server [68].

- **Decentralized federated learning:** In this type of framework there is not central server handling the protocol, but rather users exchange information with each other for model aggregation and sharing updates. As the exchange of information is handled only between users, it ensures that security concerns of having just one vulnerable point are solved. Therefore, this type of model enhances robustness by distributing the aggregation for the model training. Also, it improves privacy as the data and aggregation updates are only distributed between the participants of the model training [68].

Federated learning frameworks can also be classified by the amount of users and their role in the training process:

- **Cross-Silo federated learning:** This type of framework is formed by a small quantity of stable and reliable parties for the training. These parties are called silos and are usually institutions or organizations. These silos have usually great computational power and a reliable connection to the network where the model training takes place. This type of model has few participants, therefore trust is increased as parties are usually other trusted entities. Because of this, these kind of models have very stable environments as the silos possess strong computational assets as well as large enough datasets, which in combination, ensure a reliable training [68].

- **Cross-Device federated learning:** Cross-Device federated learning frameworks are based on a large network formed by many devices of different kinds, such as smartphones, edge devices or IoT devices. These kind of devices are computationally not as trustworthy, as their capabilities vary heavily from one another and their connection to the network is unstable, since they might only be able to connect to intermittently. This type of model works well for

gathering data from many different users, which makes the training more personalized. However, the quality of these datasets is typically poor, as they tend to be small and unreliable, therefore many of them are needed for a proper training [68].

1.4 Privacy and Security in Federated Learning

1.4.1 Main Risks and Threats

Federated learning is a new and innovative method, but because of its decentralized architecture, it is vulnerable to many types of attacks over the course of its usage. These risks may compromise user privacy or have an effect on the model's performance. Federated learning typically lacks the security guarantees found in centralized learning frameworks because it depends on training across a large number of possibly unreliable devices, each of which possesses private and inaccessible data. This gives adversaries the chance to take advantage of the configuration and target the security or performance of the model. Attackers can take advantage of many of federated learning vulnerabilities, as described in [81]. These attacks are usually classified in three groups according to their main goal, which could be to compromise the availability, confidentiality, or integrity of the system [22, 40].

Model Poisoning Attacks: These are integrity altering attacks. They refer to adversaries who directly tamper their own local model updates which are later submitted to the main server. They do this with the main purpose of altering model updates or injecting backdoors to degrade model performance.One of the most common methods to launch such an attack is by adding a fixed perturbation or replacing the true parameters of the update with constant values [81].

In 2023, Bagdasaryan et al. [8] presented the first backdoor attack on federated learning, in the context of backdoor development. They attack is made by first training a malicious model that closely resembles the behaviour of the global model, and then secretly replacing the legitimate global model update with the malicious one. In order to improve the effectiveness of this substitution, they lower the learning rate, which prolongs the persistence of the backdoor, and add an anomaly detection penalty to the loss function to get around detection systems. This method needs detailed tuning of the model parameters and works best when the global model is getting close to converging [40].

Data Poisoning Attacks: These attacks are integrity altering attacks. Data poisoning attacks are different from model poisoning attacks because they do not inject noise directly into the gradient, but, they train models with poisoned data to craft tampered gradients. After that, they use one or multiple devices to send these corrupted gradients to the main model and therefore poison the training [81].

One of the most common types of data poisoning attacks is the label flipping attack. [113] and is labelled as an invisible poisoning attack. In label flipping attacks, an adversary misaligns training data labels with the training data they are associated with, with the purpose of downgrading performance of the attacked joint model. For this particular attack type, feature collision is used, where the constructed gradients

are very similar to source updates within a latent feature space, though often often target specific instances [40].

Poisoning attack can also be visible, where a introducing a backdoor in a system requires malicious updates specifically designed for that purpose. A new backdoor attack called Chameleon was proposed by Dai et al. [27] in 2023. This attack enables adversaries to create more resilient visual backdoors with adaptability capabilities for peer-to-peer images. The persistence of the backdoor relies on having available two different sources of benign images that are identical and are strongly associated with the poisoned one, because these benign images and the poisoned image share the same tag.

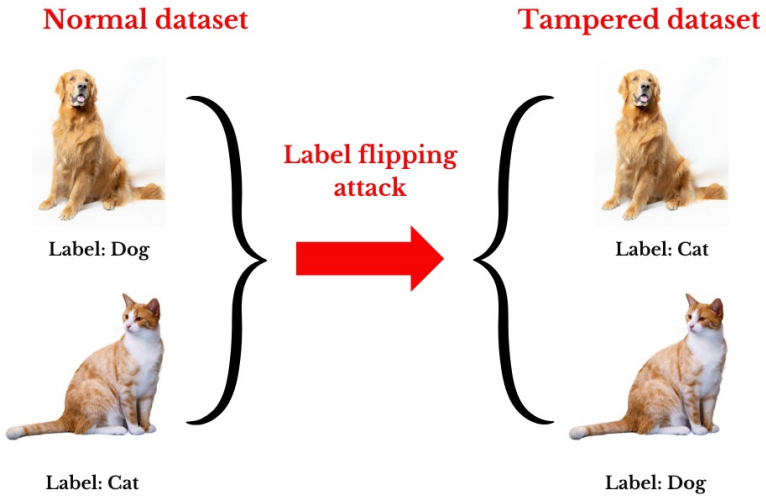


Figure 3: Label flipping attack example. "Dog" labelled samples are flipped to "cat" and vice-versa. Based on [40].

Promise Breaking Attack: They are integrity and confidentiality compromising attacks. Throughout the federated learning lifecycle, clients may agree to contribute with their training data or computing resources in exchange for participation opportunities or rewards for their contributions. Nonetheless, there can be adversary clients who could misuse the system, for example, accessing a shared model or obtaining sensitive data without performing agreed obligations [125, 41].

Though it might not be intentional to poison the model, the non-compliance with its duties from the malicious clients can lead to a lack of computational resources as well as training datasets. Also, these practices would create an unbalanced reward distribution system. Consequently, the performance of the model would be degraded, while also having ethical implications and reducing participant motivation [40].

Byzantine Attacks: They are attacks that compromise both the integrity and availability. They usually occur in distributed systems where adversaries are present and try to achieve consensus among all the nodes while using those adversary nodes.

The security and resilience of distributed architectures like federated learning systems are seriously threatened by Byzantine attacks [40].

The Sybil attack, first presented by John Douceur in 2002 [33], is one of the most well-known Byzantine attacks. The adversary creates a large number of false identities, nodes, or user accounts in this attack that seem authentic and independent but, in reality, they are all managed by the attacker. Many different strategies, including crafted IP addresses, anonymizing proxies, or virtual machines, can be used to create these false identities. Systems that rely on identity verification or trust, like peer-to-peer networks, social media platforms, and blockchain systems, are the main targets of the Sybil attack. Attackers can manipulate resources, dishonest legitimate users, and disturb consensus protocols by oversaturating the system with these fake entities. As an example, a Sybil attack in a peer-to-peer system could alter file distribution using fake nodes, leading to an imbalance in resources or decreasing the performance of the system.

Inference Attacks: Inference attacks, also called reconstruction attacks, are confidentiality related attacks. Throughout the training phase, the model will unavoidably learn inherent information from the users' data. This makes it possible for malicious clients to perform inference attacks that will recover users' original data without having any previous information about it. According to [81], there are three main types of inference attacks:

1. *Membership Inference Attack (MIA):* In federated learning applications, users or adversaries can perform MIAs to determine if a data sample is part of the training dataset.

2. *Property Inference Attacks (PIA):* Without having any direct contact with the training data, the objective of property inference attacks is to infer characteristics of this private, such as parameters or distribution.

3. *Representative Inference Attacks (RIA):* In representative inference attacks, adversaries create as output a dummy dataset that is gives a representation of the training dataset of the target.

Inversion Attacks: They are confidentiality related attacks. Inversion attacks are a type of property inference attacks, as their main purpose is using some machine learning framework to infer training data from the model. These attacks rely on the update gradients that are being updated during the federated learning model training, as it has been found that these gradients possess inherent information about the training data, and it can be revealed by inversion techniques.

The first gradient inversion algorithm for federated learning models was DLG (Deep Leakage Gradient) [144], and it specifically focuses in computer vision areas. It is based on randomly generating dummy input samples x' and labels y', that correspond to a random gradient G'. Then x', y' are iterated and optimized through the algorithm L-BFGS, that minimizes the difference between the dummy and benign gradients. The objective function for DLG is formally stated as the following formula:

$$\underset{x',y'}{argmin}||G' - G||^2.$$

Further extending the line of DLG, there have been new algorithms created that improve in high-dimensional gradient analysis and larger batch sizes, with more complex objective functions. These are algorithms such as CGL [82], GrandInversion [139] and GradViT [58].

Eavesdropping (Man-in-the-Middle Attacks): They are confidentiality related attacks. The adversaries are situated in the communication channel between central server and local users, therefore, they can launch eavesdropping attacks. The objective of the adversaries is steal or tamper some sensitive information, such as model weights or gradients, transmitted in each communication. Since in federated learning model training the server and users have frequent communication, it is inherently weak against eavesdropping attacks. The risk is especially significant when interactions occur in plaintext or uses weak encryption methods [94]. Since the exchanged data usually consists of models or gradients, attackers can further launch inference attacks to analyse the intercepted messages and extract private information.

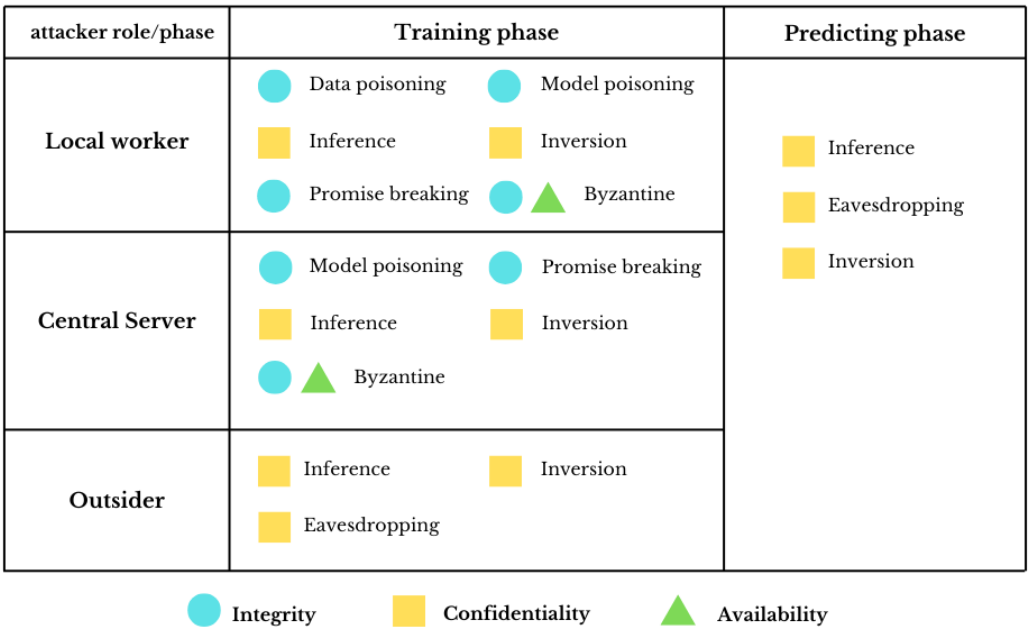


Figure 4: Summary table with attacks classified by role of the attacker and phase of the model training when they occur. Based on [22].

1.4.2 Security and Defence Mechanisms

As explored before, there is a variety of methods that adversaries can use to attack federated learning based frameworks. These range from different complexity and

have different objectives, however for every attack technique, there is a defensive method being developed. In this next section, these defensive techniques are going to be explored and explained in detail, mentioning their mathematical background, different algorithms that utilize them in real life and enumerating their advantages between the previously studied attacks. Some of these defensive methods follow different trust assumptions that will be specifically mentioned in their particular cases. Also, in the description of every technique it will be mentioned which attack or attacks they defend against from the previously mentioned group.

Data Sanitization: Data sanitization is a technique introduced in order to prevent integrity attacks like data poisoning, which attempts to eliminate adversary and questionable data prior to the training process.The study [26] explores an innovative training phase framework for FL, incorporating a data sanitization phase to it. This stage uses various subsets of the training data to produce multiple labelled micro-models in each iteration. The models assign provisional labels to their corresponding data subsets, which then undergo a voting process that helps identifying potentially malicious data.

There are also algorithms that apply different strategies for data sanitization, like FedDiv [76] which uses a global filter that is trained in parallel to the model, or like study [28],that explores data shuffling techniques to combat various attacks.

Data sanitization provides is an effective defensive technique that potentially keeps the training data clean, however its main inconvenient is that it relies entirely on accessing and examining user data, triggering privacy related issues. A novel study [71] reveals that data sanitization can only be performed successfully in cross-silo FL frameworks, that have secure authentication protocols and being executed by the users themselves. It is important to note that the study assumes clients to be completely trustworthy, something not applicable in real world FL.

Anomaly Detection: When performing attacks, adversaries follow different patters unconsciously, which left traces in the benign user data. Consequently, anomaly detection has been explored as an option to identify these suspicious patterns, using different statistical or behaviour analysis techniques. Therefore, this defensive system helps counter data crafting attacks like data poisoning. Study [23] divides anomaly detection in federated learning in two perspectives, from the client and from the data.

Client anomaly detection in FL can utilise data from the framework, which refers to user model updates, or from the communication, like IP address or user ID.

There are various approaches on how to analyse all these variables in search for suspicious patterns, for example, study [136] introduces a technique called Auto-encoder-Based Anomaly Detection that detects client model updates with unusual weights and erases any possible negative impact intended from the attackers. Also, taking inspiration from spectral anomaly detection, study [117], suggests a FL framework that tells apart abnormal data by capturing and comparing it with normal data features. This spectral anomaly detection model is applied through every learning iteration, being run by the centralized training model. The detection threshold of the system is updated through every iteration, consequently making it challenging

for the adversary to predict and understand and also, ensuring the effectiveness of the suspicious data detection.

Data anomaly detection looks for anomalous data that might have been added during the sampling or data collection process and could lower the quality of the model's training. Its main objective is finding outliers that substantially deviate from the average feature distribution. A two-phase iterative approach has been proposed by the study [105], which serves to detect adversaries by identifying samples affected by data poisoning. In a similar line, paper [70] presents a recursive auto-encoding anomaly detection method that lowers the possibility of overfitting to anomalous data. However, these techniques usually need direct access to user data, suffering from the same privacy shortcomings as data sanitization techniques.

Adversarial Training: This defensive technique is performed during the training phase, when a pre-designed and negligible perturbation is purposely introduced into the model, with the main purpose of enhancing the defences of the model in order to make it more robust. This is a defensive method used to counteract utility or availability attacks, as the purpose is to make the model fight these attacks by itself.

Adversarial training has been tested in centralized FL settings [119] and also in decentralized frameworks [77]. Study [119] applies adversarial training with a technique called ensemble adversarial training, which takes pre-trained datasets, introduces perturbations in them and then feeds them to the model as training data. Furthermore, paper [77] uses adversarial training to improve the performance of the FL model, reducing its drift and enhancing its convergence speed.

However, since introducing perturbations inevitably impacts the accuracy of the classification in the model, adversarial training may not offer the needed stability and performance needed for battling complex black-box attacks. Additionally, for a successful adversarial training, it is required to be performed in large amounts of training data, which inevitably increases the computational cost in the user's devices for training the model. This issue is especially problematic in environments with cross-device FL frameworks that have a large number of participants, where many of the users with lower resources cannot handle such high computational costs.

Model Compression: Model compression is a set of techniques applied to deep learning models, that are used to reduce their size while maintaining performance. In Federated Learning, compression techniques help to reduce the communication overhead, parameter redundancy, therefore improving efficiency. Its main purpose is to mitigate security risks linked with large model updates.

Model compression can be performed using many techniques, each one having different possible applications:

The first approach is named knowledge distillation, where, instead of sending gradients directly, clients share knowledge representations with the central server. These knowledge representations are based in probability predictions that transform the data labels into probabilities following the next formula:

$$p(y|x) = \frac{e^{z_y/T}}{\sum_j e^{z_j/T}},$$

where z_y is the output of the model for class y and T is a parameter called temperature, that smooths probability outputs and prevents overfitting.

This model compression technology has been implemented into a federated learning framework called FedMD [35]. FedMD enables users design different network structures based on their distinct computational resources. This allows to protect their data privacy and makes local models performance rise. However, this algorithm requires the users to share a part of their datasets with the central model for the training. In general, federated knowledge distillation defends against gradient inversion attacks, as there is not raw information that can be intercepted and also protects client data as only soft probabilities are share. On the other side, this technique requires more local training as the users need to train models independently.

The next technique is named pruning, which removes unnecessary or poisoned model parameters, making the model smaller while maintaining accuracy. Pruning removes these parameters following a mathematical function $P(W, \tau)$, that removes weights from gradients that are below a threshold:

$$W' = P(W, \tau) = W \cdot 1(|W| > \tau),$$

where W' is the pruned parameter, τ is the threshold and 1 is the indicator function:

$$1(|w| > \tau) = \begin{cases} 1, & \text{if} \quad |w| > \tau \\ 0, & \text{otherwise} \end{cases}.$$

There exist three types of pruning: magnitude based, that removes small magnitude weights; structured, which eliminates entire interrelated neurons, filters or layers, and unstructured, which randomly removes weights.

Pruning reduces the model size, which limits the attack surface adversaries have available, consequently reducing gradient leakage. Therefore, pruning defends well against model inversion attacks. However, one drawback from this technique is that it often requires the model to be retrained after the pruning, as many of the parameters have been deleted during the process.

Pruning has recently been studied and applied to many federated learning frameworks. For example, the research [65] introduces the PruneFL, that has an adaptive nature and prunes parameters in a distributed manner, maintaining the efficiency of the model and making the learning process shorter with more less computational needs.

The following technique is named quantization and it consists on compressing the model's weights and gradients by reducing their numerical precision. For example, instead of utilising 32-bit floating point numbers, quantizations converts them to lower bit parameters such as 16-bit, 8-bit or 1-bit.

Formally, quantization is represented in the following way. Given a weight matrix W, quantization is defined as:

$$\hat{W} = Q(W),$$

where $Q(\cdot)$ is a quantization function that converts full precision values to a chosen discrete set, depending on the precision wanted.

There are many types of quantization, depending on the discrete set the values get mapped to, or the type of quantization function they use. Some examples are:

uniform quantization, that maps values to intervals of fixed and pre set sizes. Non-Uniform quantization, that uses logarithmic or other non linear functions to map values in a determined way. Ternary quantization simply takes weights and maps the values to the set $\{-1, 0, 1\}$, and even more simplified, binary quantization maps weights to the set $\{-1, 1\}$, reducing significantly the storage space of the weights.

Quantization reduces communication costs, as the information transmitted is less complex. It also prevents inference attacks as the weight updates reduce gradient leakage and also it is harder for attackers to reconstruct original weights. However, quantization can reduce the model's accuracy, as information is being lost and this affects in performance.

This technique has also been studied and applied to federated learning recently. For example, study [75] introduces the algorithm FedFQ, that applies a fine-grained adaptative quantization strategy to FL. This algorithm allows users to customise their bit width, therefore it makes the training of the model maintain its convergence performance while also having efficient communication compression.

Finally, the last technique is top-k sparsification, which is a compression technique based on sending only the largest k% of gradients, instead of sending the totality of them for the training of the model. This is mathematically formulated in the following way:

For a vector of gradients $g = (g_1, g_2, \ldots, g_n)$, let:

$$\hat{g} = g \cdot 1(|g| \in topk\%),$$

where k is the previously set up threshold and $1(\cdot)$ is the indicator function:

$$1(|g| \in topk\%) = \begin{cases} 1, & \text{if} \quad |g| \in topk\% \\ 0, & \text{otherwise} \end{cases}.$$

This technique helps with data leakage and defends from gradient crafting attacks and model inversion. However, since the amount of gradients that are being sent for training is reduced, this might affect the convergence speed of the model.

One example of application of top-k sparsification in FL is the study [116]. This study introduces an innovative approach, combining top-k sparsification with a shuffle model, resulting in an reduction of communication overhead in the federated learning process.

Byzantine-Resilient Aggregation: Simple gradient aggregation models, like FedAvg which uses linear combinations, have been proven to be more susceptible to byzantine attacks when being used in the training of federated learning models [12]. Using just linear combinations, such as averages, just one adversary can break into the global model, corrupt the training and in some cases prevent the model from converging. To mitigate the impact of these malicious users, byzantine resilient aggregation techniques were designed. These ensure that model is robust and safe even when some clients try to corrupt it. There are many techniques suggested to replace the less secure averaging aggregation, and some of these are explained below:

The first technique is named statistical feature-based robust aggregation. When adversaries introduce some perturbations in gradients, these updates are difficult

to filter and tell apart from other normal updates in the FL process. Based on this knowledge, researchers have thought of statistical-based aggregators. These aggregators are on the server and using different statistical methods they measure who the most trustworthy users are and only use their updates to train the model, keeping away malicious updates.

The KRUM algorithm [12] is one of the most used statistical techniques to select this trusted users. KRUM selects one client update per round, which is expected to be the closest to the majority of honest clients and then selects their gradient to be used in the aggregation process. The mathematical formulation explaining how the KRUM algorithm selects the most trustworthy client is the following:

$$g* = \underset{g_i}{argmin} \sum_{j \neq i} d(g_i, g_j),$$

where $g* \in \mathcal{G}$ is the selected gradient from the set of gradients $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$ received from the n clients. KRUM computes $d(g_i, g_j) = ||g_i - g_j||_2$, the standard euclidean distance between every pair of gradients. Then, for each gradient g_i, it sums the distance to its closest $N - f - 2$ users, where f is the maximum assumed number of byzantine users. Finally, it chooses as $g*$ the gradient with the lowest total sum of distances, i.e., the gradient that is closest in average to its neighbours.

The KRUM algorithm is effective against byzantine attacks because if adversaries submit arbitrarily large or malicious updates, they will have a large pairwise distance respect to the gradients of honest users. Also, the algorithm only considers the closest $N - f - 2$ gradients, making sure byzantine gradients are ignored. However, this algorithm has a high cost of computation, as it computes pairwise distances, and slow convergence, since it only selects one gradient per iteration. Finally, its main flaw is that the algorithm can select adversarial updates if there is a majority of byzantine users.

To solve the slow convergence flaws an improved design of this algorithm called Multi-KRUM [12] was designed. Multi-KRUM has a similar structure as the basic KRUM algorithm but, in the step where KRUM selects the most trustworthy gradient, Multi-KRUM instead selects the top k gradients with the smallest sum of distances. Then the k gradients are aggregated computing the average between them, using the formula:

$$g* = \frac{1}{k} \sum_{i \in S} g_i,$$

where S is the set of the selected k best gradients.

Therefore, Multi-KRUM solves the speed convergence issue as it aggregates multiple gradients at once instead of just one, making the training much faster. Also, as Multi-KRUM averages k gradients it is less likely to be affected by one malicious gradient, however, choosing this number k can prove challenging as if it is too large, the byzantine updates could be included in the aggregation, but if it is too small the convergence can be slow. It is important to note that Multi-KRUM also has the issues of high complexity, due to the euclidean distance computing, and that it needs a minority of byzantine gradients in order to converge and guarantee robustness.

Another statistically based algorithm used in robust aggregation is the Trimmed mean and median [138]. The trimmed mean algorithm takes every independent

parameter of a gradient and sorts the values in ascending order. Then trims the bottom and top βN values, in order to eliminate outliers. Then, the average of every parameter is calculated and the vector compilation of those averages is the resultant aggregated gradient. This is represented with the following expression:

$$g^*[j] = \frac{1}{|S|} \sum_{i \in S} g_i^t[j],$$

where g^* is the resultant gradient and S is the set of trimmed gradients, i.e., the set where the top and bottom βN gradients have been excluded.

The trimmed mean is a fast and simple but effective algorithm, it removes extreme values, therefore in defends well against malicious updates. It is particularly effective when the number of byzantine users is not too large and it works well in large scale FL frameworks. The main limitation is that the parameter β has to be chosen carefully for the algorithm to be effective against byzantine users.

Now, the median simply takes the element wise median of all the gradients in the model as the aggregated gradient. It is represented by the following expression:

$$g^*[j] = median(\{g_i[j]\}_{i=1}^N).$$

This algorithm is also simple but effective, as it removes extreme values, which are often related to adversarial gradients. It is more efficient than algorithms like KRUM due to its simplicity, but it is even more vulnerable when byzantine clients are the majority. This algorithm is usually combined with the Trimmed Mean algorithm for better results and robustness.

In the real world, these algorithms are explored more deeply and usually combined in order to build stronger resilience, such as in study [90]. In this study the algorithm Bulyan is introduced, which uses the KRUM and Trimmed Mean algorithms. First, thanks to the KRUM algorithm, Bulyan first selects a poll of trustworthy gradients, which are afterwards run through the Trimmed Mean algorithm in order to generate the aggregated model update.

It is necessary to remark that all of these algorithms operate under the premise that the defender knows the quantity and distribution of adversary users, which is a challenging assumption when trying to apply these schemes to FL in real life. There have been some studies that address this limitation, for example, study [16] proposes a scheme named Sniper that does not rely on previous knowledge about the adversary and identifies non malicious model updates.

The next technique is local model performance-based robust aggregation. The main characteristic of this aggregation method is that it assigns different weights to client updates depending on the local model performance. Instead of simple averaging methods, like FedAvg, with this technique robustness is improved as the influence of adversary updates is reduced by the weighted averaging.

The aggregation process starts with an evaluation of the local model performance, which can be assessed in different ways. Every client i trains a local model w_i on its own private dataset and then reports its update g_i to the server. Some of the techniques the server can use are: validation accuracy, where each client evaluates their own model w_i on a pre agreed validation set, loss function, where the server

computes the cross-entropy loss of every w_i to compare it to the rest of models, or consistency checks, where the server compares updates g_i to previous rounds trying to detect noticeable changes. The server then assigns a performance score P_i to every user i and with that it computes the weights α_i of every client update using the following expression:

$$\alpha_i = \frac{P_i}{\sum_{j=1}^{N} P_j},$$

therefore, α_i is the normalized weight for user i based on its performance score. After assigning weights to every user update, the server aggregates all of them computing a weighted average:

$$g^* = \sum_{i=1}^{N} \alpha_i g_i,$$

with g^* being the final aggregated update of the model.

As stated, there are different methods for a server to quantify the performance score of the user models. For example, in the case of the algorithm FLTrust [17], the server keeps a small and trusted dataset, that it is used as its training dataset. This trusted dataset is called server-version model and it is normally trained by the model in every iteration. Then when the server receives the gradients from the users, it computes their cosine similarity respect to the model version the server possesses. The gradients that have the most similarity with the server-model are awarded with a higher weight for the aggregation process.

The algorithm Sageflow [99] follows a similar procedure for aggregation, but in this case the algorithm assigns the weights to the gradients after evaluating their entropy compared to the server-version dataset. Sageflow also assigns weights taking into account the volume of data that a user owns, the larger amount of data the bigger the weight.

Therefore, local model performance-based robust aggregation prevents the influence of malicious users submitting malicious gradients, can handle non IID data, as it reduces the weights of clients that submit trained updates with biased datasets, improves model convergence as it prioritizes the best quality updates and is adaptable to different metrics and functions for the evaluation process. However, this method assumes that the evaluation is honest from the hand of the users, because the performance metrics can be tampered by malicious clients. Also, this technique requires additional computations in the local level, which can be costly for some small users, and also brings a disadvantage to users that possess small datasets, as weights are sometimes awarded with lower weights.

When training a FL model, users may have different characteristics or preferences, which makes fitting all of their data updates in a single model quite challenging. This is why the next technique, clustering-based robust aggregation, was studied and developed in FL. This aggregation method improves robustness and efficiency of the model training by dividing clients into clusters based on their similarity and aggregating their updates within those clusters before sending them to the main server.

In order to divide the users into clusters, a similarity measure $S(i, j)$ between clients i and j is computed for every pair of users. There are many options for

calculating this similarity parameter, but some of the most common choices are: the cosine similarity, that uses the following standardized scalar product between user updates,

$$S(i,j) = \frac{g_i \cdot g_j}{||g_i||||g_j||},$$

or the euclidean distance between user models w_i,

$$S(i,j) = ||w_i - w_j||_2.$$

After that, users are divided into a pre-elected K number of clusters, $C_1, C_2, \ldots, C_k$ using the similarity scores. One of the most used techniques for clustering is named K-Means Clustering and relies on minimizing variance within every cluster, using the formula:

$$\underset{C_k}{argmin} \sum_{k=1}^{K} \sum_{i \in C_k} ||w_i - \mu_k||^2,$$

where $\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} w_i$ is called the centroid of the cluster.

Next, after clusters are formed, every cluster C_k proceeds to compute an internal aggregation update:

$$\tilde{g}_k = \sum_{i \in C_k} \beta_i g_i,$$

with β_i being a weight assigned to every client in the cluster based on its quality and g_i the client update. Finally, all the cluster specific updates are aggregated together, producing the final aggregated update of the global model:

$$\tilde{g} = \sum_{k=1}^{K} \gamma_k \tilde{g}_k,$$

where $\gamma_k = \frac{|C_k|}{N}$ is the coefficient that ensures every cluster contributes proportionally to their size.

Applying all this insight to FL, ClusterFL [109] was an algorithm developed to implement the clustering approach. ClusterFL uses the cosine similarity to calculate the similarity scores of the users, trying to maximize this score when creating the clusters. There exists a stricter version of this algorithm, called Byzantine-robust ClusterFL [110]. This version follows the same procedure as ClusterFL, however, when the clusters are set up, it treats the biggest cluster as benign and the rest as malicious. Therefore the malicious clusters are discarded when the aggregation process comes and the only updates used are the ones in the biggest cluster.

The non independent and identically distributed (non-IID) nature of FL and its data is usually considered as a weak point exploited for attacks, due to its large attack surface. Study [80] suggests the Mini-FL algorithm, which identifies the user features, geography and time as the main characteristics of the non-IID data, which then are used for the clustering classification of the users. Then the algorithm creates an aggregated update within every cluster and those updates are aggregated together, creating the final model aggregation, following the scheme previously mentioned.

Clustering based aggregation is advantageous when handling non-IID, as this is an aspect where many other FL algorithms struggle with. It also clearly enhances robustness and privacy and it is a very scalable method, as the hierarchical structure organizes the users. However, its main inconvenient is that this technique is not optimal if a majority of byzantine clients dominate a cluster, as their updates will still run through the system. Clustering also may introduce computing overhead, as it may happen that not all the clusters generate updates at the same time.

Instead of filtering the information received looking for malicious indicatives or trying to classify users to eliminate adversaries' influence, training function optimization-based aggregation [131] is based on the usage and optimization of the DL loss function used to determine the optimal combination of client updates. Compared to the other robust aggregation methods explained previously, this one is in early states of research still.

The data this technique uses is called server-side proxy data. This is a limited dataset that the main server maintains, which is representative of the whole data distribution. Therefore, this proxy data is used for evaluating and finding the optimal way of aggregating the client updates. This is made by assigning weights to clients, determined by the analysis of the proxy data, that are learned dynamically. This is because, the main objective of the algorithm is trying to dynamically minimize the global loss function, with formula:

$$\mathcal{L}_s(w) = \frac{1}{|\mathcal{D}_s|} \sum_{(x,y)\in\mathcal{D}_s} \ell(w, x, y),$$

where $\mathcal{D}_s$ is the server-side proxy dataset, (x, y) are input-output pairs from $\mathcal{D}_s$, $\ell(w, x, y)$ is the loss function used by the server, such as cross-entropy or mean squared error and $|\mathcal{D}_s|$ is number of samples that are in the proxy dataset.

Some examples of algorithms applied to FL that use the training function optimization technique are, study [78], which regularizes the loss function with the purpose of making the local model and the global model not deviate from each other strongly during the training process, and research [7] that introduces an algorithm utilising dynamic clip values when specific quantiles are reached.

The advantages that this technique provides is that it is robust and defends well against adversaries, while also being efficient even with large number of clients. It is also effective handling non-IID data, a feature that many FL algorithms struggle with.

Differential Privacy: It's a defensive method against privacy harming attacks. Differential privacy helps protect data sharing by introducing artificial noise to the communication. Therefore, in federated learning systems, this technique helps ensure that private information and data of clients cannot be retrieved or reconstructed fully during both the training process and user interaction phases. [81]. There are various types of differential privacy algorithms, depending on if the noise is aggregated locally or globally:

When operating in a secure server, it is possible to centrally aggregate the noise to the gradients sent by the local users. This is known as centralized differential privacy (CDP) and it ensures that privacy is preserved all the data of the users

participating in the training. The formal definition of the algorithm is the following [34]:

Definition 1. The algorithm $\mathcal{A}$ satisfies (ϵ, δ)-DP, if for any pair of adjacent datasets $D, D' \in \mathcal{D}^n$ and outputs $S \in range(\mathcal{A})$ then:

$$Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} Pr[\mathcal{A}(D') \in S] + \delta,$$

where $\epsilon > 0$ represents the level of privacy guarantee, i.e.., the degree of indistinguishability between to neighbouring datasets D, D' and $\delta \geq 0$ corresponds to the probability of failure of the privacy guarantee. When applied, the constant δ has a negligible value, of the order $\delta << 1/||D||$.

CDP for federated learning was first introduced in 2014 by Geyer et al. [48], and its main purpose is hiding local user's contributions during the training process. After the grounds were set up, many other algorithms for federated learning started incorporating differential privacy techniques, such as DP-FedAvg and DP-FedSGD [89], which extended federated learning CDP to LSTM language models with similar results to algorithms not using noise. It is proven that CDP-FL is effective in privacy protection, however, it faces some limitations in two main aspects: First, algorithms using CDP-FL rely in a large number of users to achieve convergence, therefore if there is not enough participants, the performance of the model could be hindered. Second, the main assumption for CDP-FL is that the main server is completely trustworthy, which might not hold in real-world scenarios.

The second possible differential privacy technique is local differential privacy (LDP). In this case, users are the ones introducing the differential privacy noise. Since, in this case, the data gets the noise introduced in the user's device, LDP does not need to assume a trusted server, while also defending data from potential eavesdroppers. The formal definition of LDP is the following:

Definition 2. The algorithm $\mathcal{A}$ satisfies (ϵ, δ)-DP, if for any pair of inputs d, d' and outputs $\forall t \in range(\mathcal{A})$ then:

$$Pr[\mathcal{A}(d) = t] \leq e^{\epsilon} Pr[\mathcal{A}(d') = t] + \delta,$$

where $\epsilon > 0$ and $\delta \geq 0$ represent the same variables as in the previous definition.

In early LDP-FL algorithms [121], it was required for the users add the noise locally to their model updates prior to sending them to the central server. Nevertheless, when frameworks increased their size, the scalability of this method was not reliable and it made LDP-FL unable to provide enough privacy guarantee. Therefore, there has been new studies proposing new methods for good performance when processing higher amounts of data, such as Piecewise Mechanism [142] and Hybrid mechanism [126].

CDP and LDP are designed to fight the same attacks but with different strategies. CDP is more scalable and easier to implement as it only adds noise in the main server, however, it is weaker against internal threats and need the assumption of a trusted server. On the other hand, LDP does not need any trust assumption and its strong against both internal and external attacks, but highly struggles with scalability.

Naseri et al. explore the difference between these two types of algorithms in the following study [96]. They argue that in the context of inference attacks, both LDP and CDP protect against MIA. Also, in the case of PIA, LDP do not work and CDP can defend these attacks but suffering significant model utility degradation.

Decentralized Learning Scheme: A recurrent risk for federated learning schemes is to have the training run to through a central server, as it is complicated to find a completely trustworthy server. Therefore, there are some alternatives that have been explored in order to improve privacy and battle confidentiality compromising attacks through a decentralized approach:

Blockchain is one of techniques that have been explored as a decentralized alternative [67]. In this approach, clients train models locally and submit their gradients to designated miners. Following cross-verification, the miners disclose the received gradients and store them in a candidate block. Each miner attempts a cryptographic proof after a predefined condition is met, and the first miner that successfully verifies the proof is able to publish its block. The block is then downloaded by all participants, who utilize it to update the global model. Blockchain-based federated learning algorithms are resistant to inversion attacks because the entire learning process avoids centralized gradient aggregation [81].

Smart contracts are self-executing pieces of code that represent agreements between multiple parties within a blockchain based federated learning system. Studies like [53] and [103] suggest using them to improve scalability. The efficiency of this system is demonstrated by the method in [53], which uses active smart contracts to monitor, handle, and protect data transmission automatically. Similar to this, the BAFFLE framework [103] optimizes the model aggregation and update processes by using smart contracts to divide global model updates into separate blocks.

Apart of improving defences and privacy, FL gives an incentive to users for participating in the process, which is beneficial for blockchain. The paper [101] implements a totally decentralized frameworks that uses blockchain algorithms to reward users that perform high or provide high quality data. This approach incentives voluntary contributions from the users and potentially improves model performance.

Another possibility to combat confidentiality challenges is to transfer from centralized FL to decentralized federated learning (DFL), as shown in the figure. The study [74] explores a DFL framework, where instead of sending the gradients to the main server, the users directly update them by aggregating information from their algorithm neighbours, in order to train a model that would fit better the user observations. While DFL offers an innovative solution for addressing the privacy shortcomings FL has, all the standardized algorithms work generally under the assumption that users possess reliable networks and cannot deal with overhead, which is not a reasonable assumption when applying this frameworks in the real world [81].

Homomorphic Encryption (HE): This type of encryption allows operations to be performed after it, guaranteeing that the computations with encrypted and unencrypted data have the same output. This cryptographic technique uses what are mathematically known as homomorphic functions, which make HE follow the

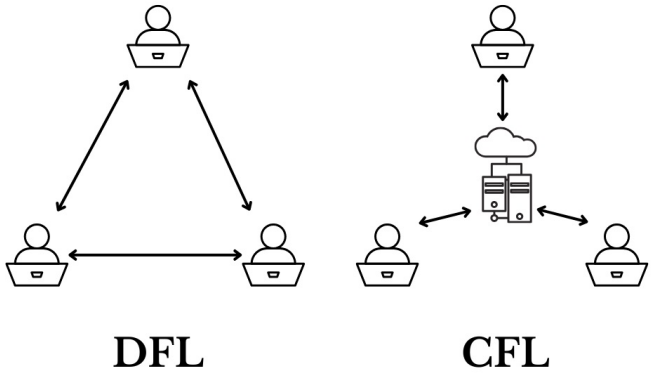


Figure 5: Figure of CFL and DFL system architectures. Based on [81].

next equation:

$$Dec[Enc(g_1) \odot Enc(g_2)] = g_1 \odot g_2,$$

where the functions, $Enc(\cdot)$ and $Dec(\cdot)$ are the encryption and decryption functions for the algorithm, and g_i is the plaintext, that in the context of federated learning is generally the model update gradient. Depending on the type of operation it uses (either addition, multiplication or both) and the limitations they have, HE can be categorized into three types:

1. *Fully HE(FHE) [73]:* Uses both addition and multiplication and their operation time is unlimited. The protection is very high, but the high computational costs limit its application to real world scenarios.

2. *Somewhat HE(SWHE) [143]:* Uses both addition and multiplication, but the operation time is limited. The operations may introduce noise during encryption, so it is applicable only to specific situations.

3. *Partially HE(PHE) [18, 36]:* It uses either addition or multiplication, but not both, with unlimited time for operations. It has a simple design but its functionality is limited, though, it can be applied across various scenarios.

Homomorphic encryption helps defend confidentiality related attacks. As the gradient is encrypted when sent to the main server, eavesdroppers are nullified. Furthermore, as operations with the data in the training of the model are being handled with encrypted data, inversion and inference attacks are prevented, assuming the adversaries do not know the function *Dec*.

The main shortcoming of HE is scalability, as federated learning frameworks often require high amounts of data processing due to the large amount of devices that participate in the model training. As mentioned, HE algorithms have a high computational cost, in particular FHE, therefore this hinders its usability for large federated learning frameworks [114].

Many HE algorithms use the Paillier cryptosystem, introduced by Pascal Paillier in 1999 [98], as its main operating algorithm. This algorithm is based in modular

arithmetics and group theory and has additive homomorphic properties. The cryptosystem makes the following equation hold for all plaintext m_1, m_2:

$$Enc(m_1)Enc(m_2) \quad mod(n^2) = Enc(m_1 + m_2) \quad mod(n),$$

which as stated, enables to operate with the data after being encrypted, using modular arithmetics.

Secure Multi-Party Computation (SMPC): SMPC was initially introduced by Yao [135], who studied the *"two millionaires problem"*, based in two millionaires that, without disclosing the exact amount of revenue they own, want to know who is the richest. Formally, given two numbers x, y representing their respective wealth, they want to learn whether $x > y, x < y$ or $x = y$, without disclosing the exact values of x, y.

The idea of secure two-party computation (2PC), which allows two parties to jointly compute a function over their private inputs without disclosing those inputs to one another, was inspired by this problem. Secure Multi-Party Computation (SMPC), which extends this concept to multiple participants in the context of federated learning, provides a method to securely carry out computations without depending on a central server. The protocol guarantees that each party only learns the computation's final result and not any additional details about the input data of other parties. This is especially useful in the model training phase, as the parameters can be input in the function by the users without revealing any data to the rest of the parties, therefore it helps combat confidentiality related attacks.

The study [13] presented PPML, a secure aggregation framework founded on a secret sharing and key agreement protocol, which incorporates SMPC into federated learning. In the algorithm, the gradients of the users are divided into several shares, which are encrypted and then combined after a certain number of them have been gathered. PPML is especially appropriate for federated learning environments, as its design does not require all the gradients from all the users to be gathered. Therefore it is not faced by stragglers, i.e., users or devices that have a later response respect to the mean and may cause a lack of performance and increase of computing time.

The protocol in formally represented using the sharing ($SS.share(\cdot)$) and reconstruction ($SS.recon(\cdot)$) processes of secret sharing, as following:

$$SS.share(s, t, U) \longrightarrow S_u, u \in U$$

$$SS.recon[(s_u), u \in U, t] \longrightarrow S,$$

where S represents the secret, u is the client that belong to the set of all participants U and t is the threshold for reconstructing S.

After this study there has been other protocol suggestions to improve certain characteristics or be applied to certain scenarios. For example, MLGuard [69] improves the robustness of secret sharing protocols for federated learning against poisoning attacks. Also, study [21] proposes an efficient and secure SMPC-FL algorithm, considering the resource constraints of edge devices.

This previous equations represent how a secret sharing scheme proceeds generally. There are many such secret sharing algorithms, but one interesting example is the

Shamir Secret Sharing(SSS) protocol [111], one of the most commonly used protocols for SMPC. It is based in polynomials and Lagrange interpolation, and its application to federated learning can be formally expressed as it follows:

1. *Setup:* The algorithm is performed by n users from the set of users U. Then if the user u wants to share secret S, the secret will be divided into n shares and will only need t, the threshold, to reconstruct it.

2. *Secret sharing function* ($SS.share(\cdot)$)*:* The secret sharing function starts with choosing a prime number p larger that S, that will set up the finite field $\mathbf{F}_p$ where the algorithm will run. The secret S is set as the constant term of a random polynomial $f(x)$ of degree $k-1$:

$$f(x) = S + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} \quad mod p \quad a_i \in \mathbf{F}_p, \forall i.$$

   Finally, n shares are computed. These shares are:

$$(x_i, y_i) = (x_i, f(x_i)) \quad \forall i = 1, 2, \ldots, n,$$

   where $x_i \neq 0$, are distinct, non-zero elements in $\mathbf{F}_p$

3. *Secret reconstruction function* ($SS.recon(\cdot)$): For the reconstruction of the secret, at least k shares are needed. Given k pairs (x_i, y_i), we can interpolate the polynomial:

$$f(x) = \sum_{i=1}^{k} y_i \cdot \ell_i(x) \quad mod p,$$

   where $\ell_i(x)$ are the Lagrange basis polynomials:

$$\ell_i(x) = \prod_{j=1, j\neq i}^{k} \frac{x - x_j}{x_i - x_j} \quad mod p.$$

   Finally, the secret is the constant term $S = f(0)$ of the interpolated polynomial. Therefore, the secret can be computed without knowing the original polynomial.

The SSS cryptographic protocol has been used as the core of many modern federated learning algorithms, for example FLAP [95], an algorithm that combines secret sharing with game theory, or UFL [20], a scheme that combines SSS with a shuffling mechanism, aiming to balance data privacy and model performance.

Trusted Execution Environments (TEE): Trusted Execution Environments' main purpose is to build protection against data and computations in untrusted remote settings. TEEs create a secure framework that ensures the privacy, integrity, and verifiability of information, protecting it from external interference. They do this by isolating specific software and hardware resources.

In Federated Learning architectures, TEEs enable models and the training phase to locally operate within a trustworthy and private network, which reduces the risk

of leaking information and makes it possible to defend against attacks such as model poisoning and promise breaking attacks [93, 24]. However, despite their security benefits, TEEs are vulnerable to side-channel attacks and they are constrained by limited memory and only have access to CPU resources. This limitation is a challenge for FL algorithms, as scalability is a recurrent issue due to its models being handled in large scale and their complex tasks often require GPU acceleration. Therefore, it is still an open challenge to integrate FL in an efficient way across secure frameworks, client appliances and edge computing devices [81].

2 Mathematical Bases

In this chapter there will be a compilation of the most important mathematical definitions, concepts and tools, needed for the construction of the cryptographic and federated learning algorithms, which are the main object of study of this thesis.

2.1 Elliptic Curves and the Discrete Logarithm Problem

Public key cryptosystems are based on the solution of a special kind of problem that is quickly solved when one special secret is known by the solver, but becomes very hard to work out when this information is not known. An example of this type of cryptosystem is the discrete logarithm problem, which is the basis of the zero-knowledge proof used the protocol that will be analysed later. Here the objective is to determine the value of x when:

$$a^x = b,$$

where a and b are given values in a chosen multiplicative group. Over ordinary integers the equation is straightforward to solve, but over elliptic curves defined over finite fields, it becomes particularly difficult.

To understand why this is the case, it is necessary to define formally what an elliptic curve is.

Definition 3. Consider the next equation, defined over a field K with characteristic different from 2 and 3:

$$y^2 = x^3 + ax + b,$$

where $a, b \in K$ are constants that satisfy $4a^3 + 27b^2 \neq 0$.

An *elliptic curve* [102] is the set of all the solutions that satisfy the above stated equation, along with another point called $\tilde{0}$, which is defined as the point at infinity. The set that contains all these points is denoted by E.

Elliptic curves have a group structure, that yields from the points of the curve and a specific inner operation represented graphically in the figure 2.1. This inner operation is the following [118] and it uses the point $\tilde{0}$ as identity element:

Let $P, Q \in E$ be two points of the curve and L the line that passes through them. The line intersects with the curve again at some point, which is denoted $-R \in E$. Then, the opposite point, R, will be the symmetric point of $-R$ with respect to the x-axis. Finally, the point $R := P + Q$, defining the inner sum operation in an elliptic curve.

It is important to note the next special situations:

- If $P = Q$, then L is the tangent line to the elliptic curve in that same point, therefore $-R = P$ is the other point where the line intersects with the curve. In this case, the sum of these points will result on $P + P = R$, or equivalently, $2P = R$.

- If the line L only intersects the curve in one point, say P, then this means $P = Q = -R$. Therefore, $P + P = -P$, or equivalently, $3P = \tilde{0}$.

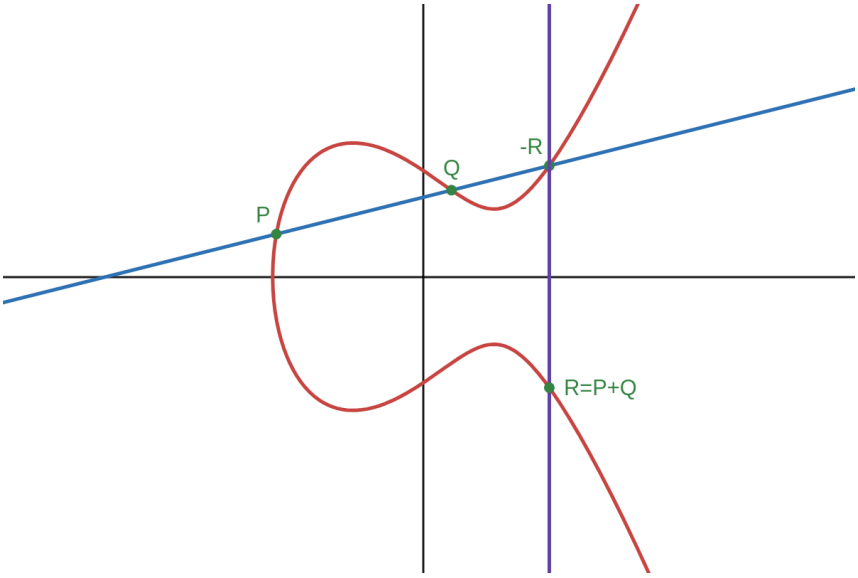


Figure 6: Graphic representation of the sum of two points in a generic elliptic curve. Figure generated with Desmos graphic calculator tool [31]

Considering the set of points in the curve E and based on the given definition of the inner sum operation, it is possible to prove that the elements of the set form an abelian group. Additionally, the next theorem named *Hasse's Theorem* [19, 45], places a boundary on the number of elements this group has:

Theorem 1. *Let E the set of points of an elliptic curve and $K = \mathbb{F}_p$ a finite field of p elements. If E is defined over $\mathbb{F}_p$ then Hasses's theorem states that:*

$$|\#E - p - 1| \leq 2\sqrt{p}.$$

Where $\#E$ is the cardinality of set E, i.e., the amount of elements the set has.

An observation about the theorem is that, the size p of the field that the curve is defined over and the number of points in the elliptic curve are very close in magnitude.

The behaviour of the discrete logarithm problem in elliptic curves is the following [45]:

Choose a large prime number p and an elliptic curve C, defined over the finite field $\mathbb{F}_p$. Let $A, B \in E$ be two points in C, such that the equation $B = kA$ holds for some $k \in \mathbb{F}_p$.

The discrete logarithm problem is based on finding the value k in the last expression. The point A is defined as generator, which is fixed, and B is defined as the hiding of k, a term that will be discussed later, with k being denoted as the private key. An important remark is that as the field's size p increases, the problem gets more complex.

2.2 Group Theory Preliminaries

Definition 4. [52, 10] Let $\mathbb{G}$ be an additive group, and let $g \neq 0$ be an element of the group. A *hiding* is a map that takes elements from a finite field, which will be

$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ in this study, and maps them accordingly to group elements from $\mathbb{G}$, ie, maps them to points on an elliptic curve:

$$\begin{array}{rcl} \mathbb{Z}/p\mathbb{Z} & \xrightarrow{h} & \mathbb{G} \\ n & \longmapsto & ng. \end{array}$$

This mapping mechanism "hides" information securely on the curve using the previously described discrete logarithm problem. In fact, computing the inverse of this mapping is equivalent to solving the discrete logarithm problem, known to be computationally challenging.

The major reason why this mapping is employed is its computational effectiveness when finding the image of any given value. Finding the hiding of n requires around $2\log_2(n)$ operations. This ratio is due to the properties of elliptic curve addition and the method of successive doublings. In particular, taking any given point and finding all its powers of two is very simple, as this type of computation is based on the special initial case discussed earlier as part of the definition of the elliptic curve addition operation.

When learning the power of two of any point, it is possible to calculate any multiple of that given point using appropriate additions. Calculating the inverse function of a hiding might require computing the function as many times as there are members in the group, as there is not a straightforward inverse mapping. Therefore, when the group has a large number of elements, this method becomes computationally inefficient.

Definition 5. [52, 91], Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two additive groups, and let $\mathbb{G}_T$ be a multiplicative group. A *pairing* is a mapping that takes a pair of elements, one from each of the first two groups, and maps them into an element in the third group:

$$\begin{array}{rcl} \mathbb{G}_1 \times \mathbb{G}_2 & \xrightarrow{e} & \mathbb{G}_T \\ (a, b) & \longmapsto & e(a, b). \end{array}$$

A pairing is bilinear if the following identities hold:

$$e(a + b, c) = e(a, c)e(b, c) \quad a, b \in \mathbb{G}_1, c \in \mathbb{G}_2$$

$$\textit{and} \quad e(a, b + c) = e(a, b)e(a, c) \quad a \in \mathbb{G}_1, b, c \in \mathbb{G}_2.$$

Also, a pairing is non-degenerate when:

$$\text{If } \tilde{0} \neq a \in \mathbb{G}_1 \text{ then, there exists } b \in \mathbb{G}_2 \text{ such that } e(a, b) \neq 1.$$

Where the neutral element of the group is denoted by $\tilde{0}$.

It is important to note that, when working with elliptic curves, there exist several types of pairings, including the Weil, Tate and Ate pairings, each having different properties. In this case, the *Weil pairing* [91] will be used for the construction of the protocol. In this type of pairing, the group $\mathbb{G}_T$ corresponds to the group of q-th roots of unity in $\overline{\mathbb{F}}_p$.

After all the tools have been described, this will be the general construction on how group operations are computed in the algorithm [52, 45]:

- Three abelian groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of order q will be used, with their respective generators being g, h, and $e(g, h)$.

- The bilinear pairing map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

- Groups $\mathbb{G}_1, \mathbb{G}_2$ should be chosen making sure that an efficient computable homomorphism between the two groups cannot exist.

- There are efficient algorithms for computing group operations, evaluating the bilinear pairing map, deciding membership of the groups, deciding equality of group elements and sampling generators of the groups. These will be referred to as generic group operations.

Thus, the algorithm follows the next diagram:

$$\begin{array}{ccccc} \mathbb{Z}/p\mathbb{Z} & & \mathbb{Z}/p\mathbb{Z} & & \mathbb{Z}/p\mathbb{Z} \\ \downarrow{\scriptstyle h_1} & & \downarrow{\scriptstyle h_2} & & \downarrow{\scriptstyle h_T} \\ \mathbb{G}_1 & \times & \mathbb{G}_2 & \xrightarrow{\quad e \quad} & \mathbb{G}_T \end{array}$$

Figure 7: Diagram of the group hiding and pairing structure

The generators of each respective group are $g, h, e(g, h)$, and the pairing and hiding maps relate following the next expression:

$$e(h_1(a), h_2(b)) = h_T(ab) \qquad a, b \in \mathbb{Z}/p\mathbb{Z}.$$

It is important to note that, this relationship between mappings enables the transformation of additions in $\mathbb{G}_1$ and $\mathbb{G}_2$ into multiplications in $\mathbb{G}_T$, as the latter is a multiplicative group. For instance, the principal kind of calculation that will be addressed later in the algorithm is based on the following statement:

$$(a_1 + \cdots + a_n), (b_1 + \cdots + b_m), (c_1 + \cdots + c_l) \in \mathbb{Z}/p\mathbb{Z}$$

$$s.t. \qquad (a_1 + \cdots + a_n)(b_1 + \cdots + b_m) = (c_1 + \cdots + c_l).$$

Where the resulting hiding structure comes from:

$$e(h_1(a_1) + \cdots + h_1(a_n), h_2(b_1) + \cdots + h_2(b_m)) = e(h_1(c_1) + \cdots + h_1(c_l), h_2(1)).$$

To respect the same notation as in the original paper of the zero knowledge protocol [52], group elements will be denoted by their discrete logarithm, represented by the hiding maps. These are, $[a]_1$ for $h_1(a)$, $[b]_2$ for $h_2(b)$ and $[c]_T$ for $h_T(c)$. With this notation the generators are $g = [1]_1$, $h = [1]_2$, $e(g, h) = [1]_T$, while the neutral elements are $[0]_1, [0]_2, [0]_T$.

2.3 ZK-SNARKs

Again, to respect the notation of the original paper [52], some notation definitions are needed for the following construction. Let A be a randomized algorithm and when it outputs a result y given an entry x, this is denoted $y \leftarrow A(x)$. Similarly, when algorithm A outputs y and algorithm B outputs z, both given the same input x and sharing the same source of randomness, this is expressed as $(y; z) \leftarrow (A||B)(x)$. Lastly, the notation $y \leftarrow S$ indicates that y is sampled uniformly from the set S.

Definition 6. [52] Let S, W be two sets, and let M be a deterministic polynomial-time algorithm that takes inputs from $S \times W$ and outputs values in the set $\{0, 1\}$. Then, a relation R is defined as:

$$R := \{(\phi, \omega) \in S \times W | M(\phi, \omega) = 1\}.$$

Elements from S are denominated *statements*, while elements from W are called *witnesses*.The primary focus of this construction is on true statements, which belong to the set:

$$S_T := \{\phi \in S | \exists \omega \in W, (\phi, \omega) \in R\}.$$

The reason behind electing elements of this specific set is that, the purpose of the algorithm is to prove that $\phi \in S_T$, while ensuring that no information is revealed about its witness pair ω. Next, there are two notable cases of statement-witness pairs [45]:

- Let $S \subseteq \mathbb{N}$ and $W \subseteq \mathbb{N}^2$. The statement will be $n \in S$ and the witness will be $(p, q) \in W$. This statement-witness pair defines the relation:

$$M(n, (p, q)) = \begin{cases} 1 & \text{if} \quad n = p \cdot q \\ 0 & \text{otherwise} \end{cases}.$$

- Let $S \subseteq \mathbb{F}^n$ and $W \subseteq \mathbb{F}^m$. The statement is $\phi = (a_1, \ldots, a_n) \in S$ and the witness is $\omega = (a_{n+1}, \ldots, a_{n+m}) \in W$. For the matrices $U, V, W \in \mathbb{M}_{l \times (n+m)}(\mathbb{F})$, the pair defines the relation:

$$M((\phi), (\omega)) = \begin{cases} 1 & \text{if} \quad U(\phi, \omega)^\intercal \circ V(\phi, \omega)^\intercal = W(\phi, \omega)^\intercal \\ 0 & \text{otherwise} \end{cases},$$

  where the operation $\circ$ is defined as, $(x_1, \ldots, x_n) \circ (y_1, \ldots, y_n) = (x_1 y_1, \ldots, x_n y_n)$

  This relation is denominated QAP and it is key in the formalization of the zero-knowledge protocol that will be later explained.

The general construction of an efficient publicly verifiable non-interactive argument for a relation R is the following quadruple of probabilistic polynomial algorithms [52]:

- $(\sigma, \tau) \leftarrow \textbf{Setup}(R)$: the setup algorithm generates a common reference string (CRS) σ and a simulation trapdoor τ for the relation R. τ is used just for theoretical purposes, in order to argument about Zero Knowledge property.

- $\pi \leftarrow \mathbf{Prove}(R, \sigma, \phi, \omega)$: the prover algorithm takes as input the CRS σ and the pair $(\phi, \omega) \in R$ and outputs the argument π. The vector π constitutes the zero-knowledge proof that will be introduced to the verifier.

- $\nu \leftarrow \mathbf{Vfy}(R, \sigma, \phi, \pi)$: the verification algorithm receives the CRS σ, a statement ϕ and an argument π and returns a verdict ν. The value of ν is 1 if the proof is accepted, and 0 otherwise.

- $\pi \leftarrow \mathbf{Sim}(R, \tau, \phi)$: the simulator algorithm uses the trapdoor τ and a statement ϕ as input, and returns an argument π. This algorithm generates a valid proof without access to the corresponding witness. Although it is not used in practice, the existence of the simulator is key when arguing about theoretical properties of the algorithm, such as Zero Knowledge.

Again, to follow the notation of the original paper's construction [52], it is important to introduce the next conditional probability notation for the upcoming definitions: $Pr\left[A_1; A_2; \ldots; A_n : B\right]$ represents the probability of B happening, knowing that $A_1, \ldots, A_n$ have occurred.

Definition 7. [52] Completeness states that, given any true statement, an honest prover should always succeed in convincing an honest verifier. A quadruple of algorithms (Setup, Prove, Vfy, Sim) satisfies **Perfect Completeness**, if for every pair $(\phi, \omega) \in R$, then:

$$Pr\left[(\sigma, \tau) \leftarrow \mathbf{Setup}(R); \pi \leftarrow \mathbf{Prove}(R, \sigma, \phi, \omega) : \mathbf{Vfy}(R, \sigma, \phi, \pi) = 1\right] = 1.$$

Definition 8. [52] An argument is zero-knowledge if it does not leak any information apart of the truth of the statement that is being proved. A quadruple of algorithms (Setup, Prove, Vfy, Sim) satisfies **Perfect Zero-Knowledge**, if for every pair $(\phi, \omega) \in R$ and all adversary algorithms A, then:

$$Pr\left[(\sigma, \tau) \leftarrow \mathbf{Setup}(R); \pi \leftarrow \mathbf{Prove}(R, \sigma, \phi, \omega) : A(R, \sigma, \tau, \pi) = 1\right] =$$

$$Pr\left[(\sigma, \tau) \leftarrow \mathbf{Setup}(R); \pi \leftarrow \mathbf{Sim}(R, \tau, \phi) : A(R, \sigma, \tau, \pi) = 1\right].$$

Definition 9. [52] Soundness states that it is not possible to prove a false statement, i.e, convince the verifier without a witness. To strengthen the concept of soundness, a quadruple of algorithms (Setup, Prove, Vfy, Sim) is denoted as an argument of knowledge, if there is an extractor that can compute a witness whenever the adversary produces a valid argument. The extractor is a polynomial-time algorithm with full access to the internal computations of the adversary that, by applying a series of simple operations, is able to obtain the witness. Therefore, the quadruple satisfies **Computational Knowledge Soundness for security parameter** ϵ if for all polynomial time adversaries A there exists a polynomial time extractor χ_A such that:

$$Pr[(\sigma, \tau) \leftarrow \mathbf{Setup}(R); ((\phi, \pi); \omega) \leftarrow (A||\chi_A)(R, \sigma) : (\phi, \omega) \notin R$$

$$and \quad \mathbf{Vfy}(R, \sigma, \phi, \pi) = 1] < \epsilon.$$

The error parameter ϵ is specific to every relation and can be made arbitrarily small. Its magnitude depends on the size of the elliptic curve chosen, the larger the curve, the smaller the value of ϵ.

After the previous algorithms and properties, it is possible to define:

Definition 10. [11, 52] The tuple $(\mathbf{Setup}, \mathbf{Prove}, \mathbf{Vfy}, \mathbf{Sim})$ is a **perfect non-interactive argument of knowledge** for R if has perfect completeness, perfect zero-knowledge and computational knowledge soundness as defined above.

To conclude, the previous definition can be generalized to include the notion of **SNARK** [11](succinct non-interactive argument of knowledge), which refer to the arguments meeting the previously mentioned requirements, while also being succinct. This means that, the proofs it produces are characterized by their shortness and also by the simplicity with which these can be verified.

3 Zero-Knowledge Algorithm: Groth16

This chapter will extend on the Groth16 [52] algorithm, created by Jens Groth in 2016. Groth16 is based in a ZK-SNARK scheme and its security relies on the discrete logarithm problem based on elliptic curve cryptography.

The Groth16 algorithm originated as an optimization of a previously created algorithm, Pinocchio [100]. The start of Zero Knowledge proofs dates back to 1985, when the study [49] was published. This paper, named "The Knowledge Complexity of Interactive Proof Systems", defined notions such as Zero-Knowledge or interactive proofs, and started building on those concepts, showing possible examples of their usage. After the publication, throughout the 90's and 2000's, the research in the area kept growing, but no real life application was developed. Finally, in 2013, the Pinocchio [100] algorithm was created, being the first efficient interactive proof based on Zero-Knowledge cryptography, that served to verify statements using general computations.

Pinocchio was the first implementation that was complete and universal for any type of computation, in the field of Zero-Knowledge. But being the first one meant there was room for improvement and soon the community started trying to polish this algorithm to make it more efficient. As a result of this effort, Groth16 was one of the more efficient improved versions of the Pinocchio algorithm. The performance was significantly improved, as to generate a proof with the Groth16 algorithm it is only necessary to use three elliptic curve points and three pairings, instead of the eight points and twelve pairings that Pinocchio uses.

Nowadays, Groth16 is one of the most stablish and standardised zero knowledge algorithms and it is used in combination with other techniques to provide security in different cryptographic protocols. In the case of this study, the main objective is the analysis of the protocol named ZKPoT [42], that will be explained in the next chapter, which uses a ZK-SNARK structure based on Groth16. Therefore, this chapter is going to explore further into the mathematical basis of this ZK-SNARK structure, supported by the theoretical concepts and definitions presented in the previous chapter.

3.1 Quadratic Arithmetic Programs

ZK-SNARKs use different methods to be able to represent the relation R and to make computations, such as: binary arithmetic circuits [43], low-depth circuits [124] or R1CS [97].

In the case of the Groth16 [52], the way to encode the relation R is using Quadratic Arithmetic Programs (QAP's). During the definition of relation, QAPs were already seen as an example of what a relation is, using two vectors of values to define one. In general, a QAP is written as the following system of equations:

$$\begin{aligned}(a_1u_{11}+\cdots+a_mu_{1m})(a_1v_{11}+\cdots+a_mv_{1m}) &= (a_1w_{11}+\cdots+a_mw_{1m})\\ \vdots &\qquad\qquad \vdots\\ (a_1u_{n1}+\cdots+a_mu_{nm})(a_1v_{n1}+\cdots+a_mv_{nm}) &= (a_1w_{n1}+\cdots+a_mw_{nm}),\end{aligned}$$

where $u_{ij}, v_{ij}, w_{ij}, a_i \in \mathbb{F}$. In this system, the values, u_{ij}, v_{ij}, w_{ij} are the coefficients and a_i are the variables.

It has been demonstrated that any problem that NP solvable can be represented as a Quadratic Arithmetic Program (QAP) [47]. This proves the universality of the protocol, meaning that the QAP framework is powerful enough to represent any NP relation.

In order to build a relation using a QAP, first it is necessary to choose a fixed length l, used in the system in the following way:

$$((a_1, \ldots, a_l), (a_{l+1}, \ldots, a_m)) \in R \longleftrightarrow a_1, \ldots, a_m \quad \text{is solution of the QAP.}$$

When applied, the values $(a_1, \ldots, a_l)$ will be public, and the values $(a_{l+1}, \ldots, a_m)$, private. Given the known public values, the goal of the QAP is to solve the system for the private values.

After expressing the relation as this specific system of equation, the QAP procedure is the following [52]:

From the n equations in the system, choose arbitrary values $r_1, \ldots, r_n \in \mathbb{F}$ and then define $T \in \mathbb{F}[X]$, the polynomial satisfying,

$$T(X) = \prod_{j=1}^{n} (X - r_j).$$

Next, define $U_i(x), V_i(x), W_i(x) \in \mathbb{F}[X]$ as the degree $n-1$ Lagrange interpolating polynomials of the i-th column, i.e., polynomials such that,

$$U_i(r_j) = u_{i,j} \quad V_i(r_j) = v_{i,j} \quad W_i(r_j) = w_{i,j} \qquad for \quad i = 0, \ldots, m, \quad q = 1, \ldots, n.$$

The coefficients of the system of equations are obtained by evaluating the polynomials in the nodes r_j.

Then, the vector of values $(a_1, \ldots, a_m) \in \mathbb{F}^m$ is a solution for the system of equations if and only if the next expression holds when evaluating the polynomials in every node $r_1, \ldots, r_n$:

$$\sum_{i=0}^{m} a_i U_i(r_j) \sum_{i=0}^{m} a_i V_i(r_j) = \sum_{i=0}^{m} a_i W_i(r_j).$$

Since $T(X)$ generates the ideal ring of $\mathbb{F}[X]$ of the polynomials that have any of the points $r_1, \ldots, r_n$ as root, then, it is also the smallest polynomial that has all the values $r_1, \ldots, r_n$ as roots. Knowing this, the previous expression can be reformulated as:

$$\sum_{i=0}^{m} a_i U_i(X) \sum_{i=0}^{m} a_i V_i(X) \equiv \sum_{i=0}^{m} a_i W_i(X) \quad mod(T(X)).$$

Analogously, there exists $H(X)$ a polynomial of degree $n+1$ such that satisfies:

$$\sum_{i=0}^{m} a_i U_i(X) \sum_{i=0}^{m} a_i V_i(X) = \sum_{i=0}^{m} a_i W_i(X) + H(X)T(X). \tag{1}$$

3.2 ZK-SNARKS for QAP's

This section will conclude with a description of how the Groth16 [52] protocol uses QAPs to construct a pairing-based ZK-SNARK [45].

The relations considered for the algorithm have the next general structure:

Let p be a large prime number, that is used to determine the size of the finite field $\mathbb{Z}/p\mathbb{Z}$. Then, the language of statements is vectors $(a_1, \ldots, a_l) \in (\mathbb{Z}/p\mathbb{Z})^l$ and witnesses $(a_{l+1}, \ldots, a_m) \in (\mathbb{Z}/p\mathbb{Z})^{m-l}$, which hold for the expression:

$$\sum_{i=0}^{m} a_i U_i(X) \sum_{i=0}^{m} a_i V_i(X) = \sum_{i=0}^{m} a_i W_i(X) + H(X)T(X),$$

where $a_0 = 1$, for some quotient polynomial $H(X)$ of degree $n-2$.

This is the definition of a standard QAP structure constructed over a field $\mathbb{F} = \mathbb{Z}/p\mathbb{Z}$. Knowing a solution for the QAP and proving it, is the same as knowing the values $a_1, \ldots, a_m$ and the polynomial $H(X)$.

As explained in the mathematical preliminaries, for the construction of the protocol [52], there are three groups of order q needed, $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, each with respective hidings $[\cdot]_1, [\cdot]_2, [\cdot]_T$, and the relations between them are expressed by a bilinear, non-degenerate pairing map.

The group $\mathbb{G}_1$ is chosen in order to have simpler representation of group elements, therefore, in the algorithm construction A and C will belong to the first group and B will belong to the second group, so that the efficiency of the algorithm is maximized.

Additionally, remember that $(a_1, \ldots, a_l)$ are public values, and $(a_{l+1}, \ldots, a_m)$ are private values, only known by the prover.

After setting up all the conditions, this is the implementation schema of Groth16 [52, 45]:

- **Setup**(R): The Setup takes as input the relation R and produces as output (σ, τ), the reference chain and the trapdoor. The elements $\alpha, \beta, \gamma, \delta, x$ are sampled randomly from the group $(\mathbb{Z}/p\mathbb{Z})^*$. The trapdoor is denoted as $\tau = (\alpha, \beta, \gamma, \delta, x)$ and after the setup, the values $\sigma = ([\sigma_1]_1, [\sigma_2]_2)$ are computed, where:

$$\sigma_1 = \left(\alpha, \beta, \delta, \{x^i\}_{i=0}^{n-1}, \left\{ \frac{\beta U_i(x) + \alpha V_i(x) + W_i(x)}{\gamma} \right\}_{i=0}^{l}, \right.$$
$$\left. \left\{ \frac{\beta U_i(x) + \alpha V_i(x) + W_i(x)}{\delta} \right\}_{i=l+1}^{m}, \left\{ \frac{x^i T(x)}{\delta} \right\}_{i=0}^{n-2} \right)$$
$$\sigma_2 = \left(\beta, \gamma, \delta, \{x^i\}_{i=0}^{n-1} \right).$$

- **Prove**(R,$\sigma, a_1, \ldots, a_m$): The Prove algorithm takes the relation R, the reference chain σ and the values $a_1, \ldots, a_m$ from the system of equations and then produces the argument π.

  The elements r, s are randomly sampled from the group $\mathbb{Z}/p\mathbb{Z}$, and next the argument $\pi = \left([A]_1, [C]_1, [B]_2 \right)$ is computed, where:

$$A = \alpha + \sum_{i=0}^{m} a_i U_i(x) + r\delta \qquad B = \beta + \sum_{i=0}^{m} a_i V_i(x) + s\delta$$

$$C = \frac{\sum_{i=l+1}^{m} a_i(\beta U_i(x) + \alpha V_i(x) + W_i(x)) + H(x)T(x)}{\delta} + As + Br - rs\delta.$$

- **Vfy**(R,$\sigma, a_1, \ldots, a_l, \pi$): The Verify algorithm takes the relation R, the reference chain σ, the values $a_1, \ldots, a_m$ from the system of equations and the argument π and produces the verdict ν.

  In the last step of the protocol, the proof is accepted by the verifier if and only if the following equality holds true:

$$[A]_1 \, [B]_2 = [\alpha]_1 \, [\beta]_2 + \sum_{i=0}^{l} a_i \left[\frac{\beta U_i(x) + \alpha V_i(x) + W_i(x)}{\gamma} \right]_1 [\gamma]_2 + [C]_1 \, [\delta]_2 \, .$$

- **Sim**(R,$\tau, a_1, \ldots, a_l$): The Simulator takes as input the relation R, the reference chain σ and the values $a_1, \ldots, a_m$ from the system of equations and then produces the argument π.

  Finally, the simulator algorithm picks values A, B randomly from the group $\mathbb{Z}_p$ and computes a completely simulated proof $\pi = ([A]_1, [C]_1, [B]_2)$. This algorithm chooses the value of C using the expression:

$$C = \frac{AB - \alpha\beta - \sum_{i=0}^{l} a_i(\beta U_i(x) + \alpha V_i(x) + W_i(x))}{\delta}.$$

This is the full schema of the Groth16 protocol. Now, to check that the presented protocol truly yields a ZK-SNARK as defined in the mathematical preliminaries, it needs to be proven that the protocol satisfies the properties of perfect completeness, perfect zero-knowledge and computational knowledge soundness for security parameter ϵ. These demonstrations will not be discussed in this dissertation but they are compiled in the original paper [52] and in the thesis [45].

4 Federated Learning Algorithm: ZKPoT

ZKPoT [42] is a decentralized federated learning algorithm that leverages blockchain structure in combination with ZK-SNARKs based on the algorithm Groth16 [52] to allow distributed data training. This section will explore in depth the ZKPoT algorithm, explaining step by step its implementation and relation with ZK-SNARKs and previous technology explained previously in the thesis.

4.1 Algorithm Overview

The ZKPoT algorithm [42] is a blockchain based framework that has a consensus mechanism. As the system is decentralized the consensus mechanism chooses a node every iteration that will aggregate all the model updates. This consensus procedure will be explained in more detail later in the description of the scheme of the algorithm.

ZKPoT introduces some innovations relative to other previous blockchain based frameworks, for example, the usage of zero knowledge proofs and ZK-SNARKs as a way of ensuring data privacy, instead of differential privacy. For example, studies [79, 84] introduced similar consensus based models, that incorporated DP to protect the models of the parameter from adversaries. This method enhances data privacy but it comes at the cost of being much more expensive computationally, therefore reflecting in the performance of those frameworks.

One of the predecessor algorithms of ZKPoT is Zero-Knowledge PoL (ZPoL) [141]. This framework introduces a combination of consensus mechanism with blockchain and ZKPs, in particular, zero knowledge convolutional networks (zkCCN) [83]. Instead of having a federated learning system for collaborative training and aggregation, ZPoL has a more traditional blockchain approach, where miners individually try to achieve a certain accuracy score to be able to write their models in a new block. This approach though is not efficient, as to verify the highest accuracy model requires up to 1000 interactions between the miner and verifier due to the statistical nature of ZKPs. Another shortcoming of the model is that the verifier is chosen at random between all the nodes of the framework, therefore making it possible for adversaries to be chosen as verifiers and tampering the course of the protocol. With the introduction of federated learning, the algorithm ZKPoT would solve this issues.

A challenge in the community is still to find the balance between security and efficiency in these blockchain based FL algorithms. To address this problem, there has been recently a trend on integrating ZKPs into FL frameworks, in order to improve privacy and robustness. Other frameworks such as zkFL [127], ZKP-FL [132], or zkFDL [5] have implemented ZKPs to protect the global secure aggregation. In these frameworks, it is the central server the one that generates the ZKPs, so that the users can locally verify the aggregation has been completed correctly. However, something that these frameworks did not fully address is that information leaks might happen when the local users are sending the model updates to an untrusted central server. It is important to know in this matter that, despite generating a correct proof of aggregation, leaking local model parameters during

the interaction could happen. Therefore, the ZKPoT framework combats this shortcomings introducing the consensus mechanism, that allows the users to collectively select a trusted aggregation node based on model performance. This solves both the computational cost of DP and the possibility of information leakage of other zero knowledge frameworks, making the ZKPoT framework secure, scalable and efficient.

An important element of many blockchain based frameworks is the InterPlanetary File System (IPFS) [9]. IPFS is a decentralized, P2P system built for more efficient file sharing and improving their storage security. The system does so assigning every file a cryptographic hash based on its content, and not based on the location, like the traditional HTTP based systems. This unique cryptographic hash the files receive is based on a Distributed Hash Table (DHT), and it changes whenever the file does, therefore making it secure against file tampering and avoiding redundancy. Files are stored between multiple nodes and they are shared between the users of the network, which makes communication faster. The ZKPoT framework uses IPFS to collect the global model aggregation ZKPs of the algorithm and store them inside its blockchain structure, which reduces storage costs and ensures integrity.

The authors state in the paper [42] that, the main goals of combining blockchain FL, with ZKPs and supporting it with the consensus mechanism are:

- Achieve a decentralized training framework that bypasses the security issues of having an untrusted server.
- Making sure the security of the local models keeps intact without compromising the efficiency and performance of the global model.
- Construct a solid leader selection process that reduces computational costs for users and is efficient for the global model.
- Ensuring scalability, making sure it is possible to construct a framework that can host a large number of blockchain nodes.

4.2 Algorithm Scheme and Implementation

As it can be seen in the figure 4.2, the ZKPoT [42] algorithm starts when the task publisher initializes the first blockchain node, publishing the IPFS address of the first global model, next to a FL tasks for the users (Step 1). Then, clients interested in the FL task can register using the blockchain structure and inspect the newest block added, where they can download the model from. After, they can start training their own local models with their private data (Step 2). Following local training, each user uses the test dataset supplied by the task publisher to determine the accuracy acc_i of their local model ω_i. Every user then creates a zero-knowledge proof (ZKP) π_{acc} matching the stated accuracy (Step 3). The IPFS addresses of the local model and the ZK proof, as well as any other information needed for verification, are then included in a transaction that the client sends to the transaction pool. The accuracy of these transactions is then sorted, with verification starting from the transaction with the highest reported accuracy. The verification goes on until a threshold number of users have confirmed that the proof is valid. Then, the user with

the highest validated accuracy is selected as the leader for that training round (Step 4). The assigned leader is then in charge of verifying the rest of the proofs that have been submitted and ensuring that the reported accuracy values meet the necessary threshold to include them in the aggregation process. When all the transactions are successfully verified, the leader creates the new global model aggregating all the local models together (Step 5). Finally, the leader stores this new global model in a new node of the blockchain, storing there the IPFS address of the new global model as well as the Merkle root of local model transactions previously used for the aggregation. Then, the blockchain includes this new block and all the clients are notified so the next iteration of the protocol can proceed (Step 6).

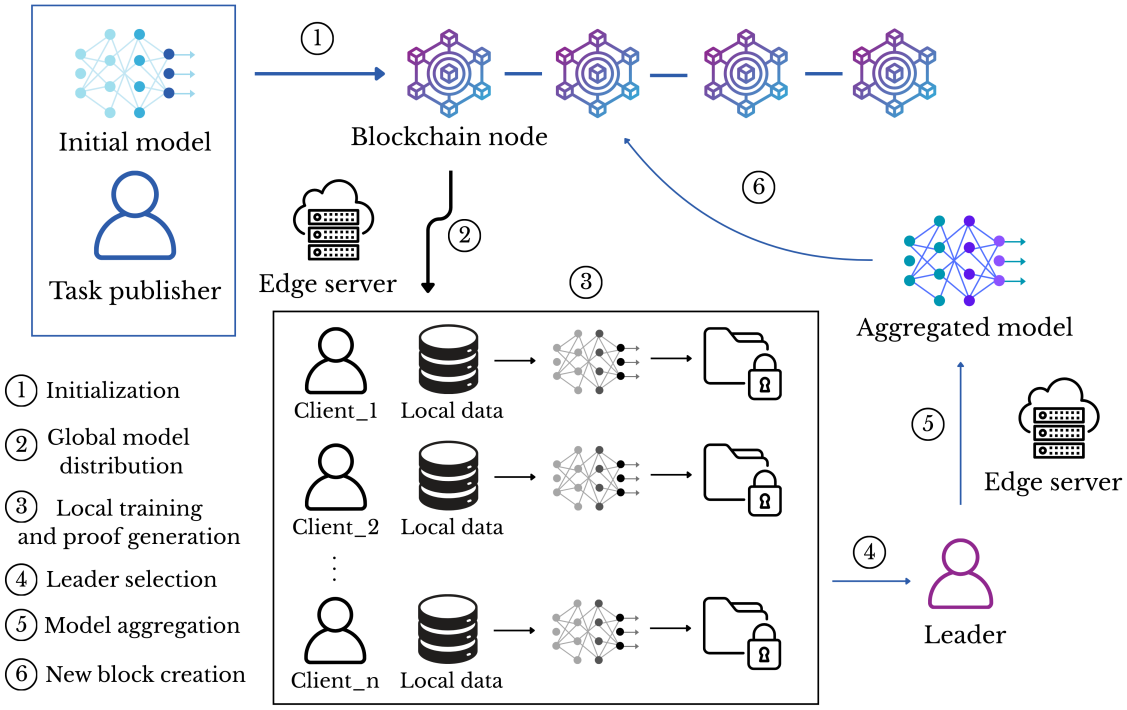


Figure 8: Graphical representation of the architecture of ZKPoT-based Blockchain FL. Based on [42].

After a general overview of the scheme of the algorithm is important to dive more in depth into the different parts of the protocol, to properly understand the construction of it and the role that zero knowledge proofs play.

The task publisher is considered to be "semi-honest", which means that it performs with reliability the setup of the ZK-SNARK protocol, between other fundamental tasks, but it is considered already that the task publisher could be interested in checking sensitive data. The main information the task publisher could be interested in is the parameters of the local models of the rest of the clients, however, these are completely impossible to know by the task publisher, unless it gets chosen as leader after participating in the training.

Another important matter when initializing the protocol is preventing bias and tampered data for the training phase. That is why the initial node only possesses the FL task, the IPFS address of the global model and some other values needed for the training. The objective of this is not letting the users know the training data in this step, preventing them to train their models directly with them and ensuring unbiased training.

After initializing the protocol, the clients obtain the initial model from the block and they start the training. In this system, the newest block has the largest height, therefore clients only need to look at this parameter to notice which is the most recent block and obtain the latest global model update W_{global}^{t-1}. With the global model W_{global}^{t-1}, the clients can train their local model ω_{global}^{t} using their private data.

It is also extremely important to note that the ZK-SNARK systems only operate with integer numbers, as they are constructed over finite fields. Therefore, it is key to perform quantization to the data, i.e., as explained before in this paper, converting floating-point numbers into simpler representations, in this case integers. The quantization used in ZKPoT is an affine map [42] that maps integer numbers q to real numbers r, following the next expression:

$$r = S(q - Z),$$

where S is a scale parameter and Z zero points.

Quantization is framed into a matricial structure to facilitate inference with quantized models. Furthermore, in order to simplify computations in the finite field, the protocol introduces two techniques optimized for R1CS and QAPs: sign-bit grouping and remainder-based verification, according to the paper [39]. These techniques are used to reduce the number of computationally demanding computations that normally appear in zero knowledge proofs, when being generated or verified.

After the quantization process is done, the clients proceed to the proof generation phase, when they demonstrate the performance of their models. Here is where the ZKPs are needed to ensure security and privacy are preserved during the steps of leader selection and model aggregation. In the case of ZKPoT [42] these proves are built leveraging the ZK-SNARK system Groth16 [52], that has already been studied in this paper. Groth16 allows the clients to generate a proof that allows the verification of a certain accuracy level the user claims to have, without disclosing any information about their models.

As explained before, the ZK-SNARK procedure starts with the setup phase, where Groth16 generates randomly the common reference string (CRS), which in this particular case includes both the proving key pk and verifying key vk. In this framework, the task publisher is the trusted user responsible for handling simultaneously two setups: of the main model globally, and of the training processes of all the users locally, bringing efficiency to this step. It is important to note again that the parameters generated in the setup are random and do not have any correlation with the parameters of the models.

Next, there is a particular step characteristic from the ZKPoT structure that differs from the general Groth16 scheme explained in the previous chapter. This is that, after the training and quantization of the local model, the client must commit

the quantized model Q into a commitment cm. This commitment ensures that: the hiding of model Q, i.e., making sure no information about it is disclosed, and the binding of Q, i.e., protecting Q from being modified without detecting it.

The commitment scheme used in ZKPoT is the Pedersen commitment scheme, which states that, for a quantized local update, the commitment of a user is computed as:

$$cm = g^{\omega_i} \cdot h^{r_i},$$

where g, h are known generators and r_i is a factor of randomness sampled by the own client. When the commitment is generated, the client sends it to the task publisher. When validated, the task publisher sends to the client D the testing dataset and T, the truth labels of the dataset. Afterwards, the client uses the dataset D for the inference process, using the matricial framework prepared in the quantization section. During the inference procedure, the clients have to document every step and record all the parameters generated in them. After the inference process is finished, the result will be the set of predictions Y, which the client will compare with the truth labels T, finally obtaining the accuracy acc of the local model.

When the accuracy is computed, the user starts the procedure of transforming the information gathered into a format suitable for computations for the cryptographic verification. This requirement is met when the information is converted into a Rank-1 Constraint System (R1CS), a system of linear equations expressed as constraints, in order to convert information into the QAP structure that Groth16 needs for operations in its ZK-SNARK. The client uses pk to buld a cryptographic proof that will satisfy these constraints and will represent the information that they want to proof. When this constraint system is built and the proof is generated, any user in the blockchain network that possesses the verification key vk can use this key to confirm that the accuracy of the model is in fact what the user stated.

Based on the previous steps, the ZKP structure of ZKPoT is defined as the next set of algorithms:

- $(pk, vk) \leftarrow \mathit{ZKPoT.Setup}(1^\lambda, Q)$: The task publisher randomly creates the public key pk and the verification key vk, while utilising as input a security parameter λ and the quantized global model Q.

- $cm \leftarrow \mathit{ZKPoT.Commit}(Q, r)$: Given a randomness factor r, the client generating the proof commits to their local model Q, giving as output the commitment cm.

- $(acc, \pi_{acc}) \leftarrow \mathit{ZKPoT.Prove}(pk, D, T, Q)$: The prover conducts the inference process on the testing dataset D. When finished, compares the predictions Y to the truth labels T obtaining the accuracy acc of the model and generating the corresponding proof π_{acc}.

- $\{0, 1\} \leftarrow \mathit{ZKPoT.Verify}(vk, cm, D, T, acc, \pi_{acc})$: Using as input the verification key vk and the proof π_{acc}, the algorithm verifies if the next statements are true: cm is a commitment for Q and, the number of correct predictions on dataset D using model Q is acc.

4.2.1 Security of ZKPoT

After analysing how the ZKPoT algorithm functions it is important also to overview why this algorithm ensures security for the users and specially for the transmission of information.

During the setup phase in the Groth16 ZK-SNARK, the security parameter λ determines the cryptographic strength of the elliptic curves and size of the finite fields used for the computations. The higher λ is, the bigger the curve and the larger the field. Therefore, this parameter can be modified depending on the level of complexity wanted for the cryptographic system. Following this, the CRS generated in the setup is totally independent of any parameter of the quantized model Q, however, the CRS always matches the size and type of system needed for generating a proof for Q. The generators g_1, g_2 that the Groth16 algorithm uses for the hidings in the groups G_1, G_2 are also chosen without any type of bias relating the model Q. Furthermore, the elements $\alpha, \beta, \gamma, \delta$ from the Groth16 CRS are randomly generated and only influenced by the value of λ. All these aspects ensure that the construction of the CRS and the setup phase of the algorithm completely satisfy the zero-knowledge and integrity properties, required for a functioning and solid ZK-SNARK, making the algorithm protect all the sensitive information the model Q might have.

The next step of the ZK-SNARK protocol is the commitment of the local model, that later is shared for verification. The client commits their local model ω_i and introduce it as a coefficient in the commitment formula, $cm = g^{\omega_i} h_i^r$. In this formula, the coefficient r is chosen at random and it is the parameter that ensures the information about the model ω_i is kept hidden. This makes sure the commitment's randomness is uniform over the group generated by g and h. It should be noted that h is chosen in such a way that its discrete logarithm with respect to g, $\log_g h$, is unknown. This guarantees that any information regarding ω or r cannot be computationally extracted from the commitment equation. Consequently, making the commitment completely secure against model information leaks to malicious users.

Finally, the algorithms *ZKPoT.Prove()* and *ZKPoT.Verify()* are in charge of the proving and verification processes. For the algorithm *ZKPoT.Prove()* to produce the proof π_{acc}, the user must have a local quantized model Q with accuracy acc on dataset D. As shown earlier in the ZK-SNARK and Groth16 analysis, the zero-knowledge property ensures that the proof π_{acc} does not disclose any information about the model and, specifically, about the veracity of the statement. This implies also that, all the steps that form this protocol and that make able to verify the zero-knowledge proof, truly satisfy the ZK-SNARK properties and ensure complete security about the disclosure of model parameters.

After analysing the security of the ZKP itself, it is significant to note the consensus achievement procedure, which adds another layer of trust to the algorithm, as it avoids the usage of possibly unreliable and untrusted central servers. This consensus achievement method for choosing a leader for every aggregation rounds functions the following way:

All the clients participating in the training have to generate a proof and make a transaction inside the server that should contain the IPFS location of their model Q, the proof π_{acc}, the commitment cm, and the accuracy acc. This transaction is sub-

mitted to the pool, that will accept transactions until reaching a certain threshold of submissions, reaching a deadline or having all the users send their transactions. When finishing the transaction round, these are sorted based on accuracy by the task publisher, and afterwards, the whole client network starts verifying the proofs utilising the *ZKPoT.Verify()* algorithm. As the transactions were classified by accuracy, the users will start verifying the highest accuracy proofs first. Whenever a proof has been verified and approved by two-thirds of the users, the client that sent that proof is selected as leader. The next step for the leader is to finish verifying all the transactions and generating a new one that contains all the IPFS directions of the models that have been verified. Next, the leader will aggregate all the local models together to generate the new global model W^i_{global}. The leader does so leveraging the FedAvg [88] algorithm, the standard algorithm used for federated learning aggregation. Lastly, a new blockchain node created by the leader and added to the chain. This block contains the new global model's IPFS location and all the verified transactions realized during the training round.

To back up all this security claims, the authors perform a performance analysis in the paper [42], using a Python and Rust based implementation of the algorithm. In the paper they mention more technical aspects about the implementation, such as the elliptic curve used for the Groth16 algorithm, the python library used in the implementation, or the dataset used for the training.

To analyse the privacy of ZKPoT the authors perform two attacks on the implementation, a membership inference attack and a model inversion attack [42]. For the first attack, the goal is to discover if a given data sample has been naturally created with the original data or artificially crafted by the attackers. For the experiment, the attackers estimate the distribution of the malicious datasets utilising the target model gradients' distributions. The inference attack takes part during the verification phase of the algorithm and its effectiveness is measured creating and evaluating a binary classifier that distinguishes original and crafted training data. The result and analysis of this attack proves that the security approach of ZKPoT truly ensures security, as attempting to model the training data is infeasible and the only way of doing it is by guessing randomly.

In the second attack, the adversaries try to reconstruct the training dataset of the target with inversion techniques. The attack is designed in a white-box scenario, where the adversaries have complete access to the model parameters of the target. The adversary starts the attack generating a dummy dataset that is optimized after each iteration to maximize the cosine similarity between the crafted and real gradient. To study the effectiveness of the attack, the authors use the inversion influence function (IF^2) [42, 140], where a lower score shows a better reconstruction of the data. In the study it can be seen that the IF^2 score is high and unchanged during every iteration. This is because the ZKPoT algorithm completely restricts users that are not verified from obtaining access to other models, therefore the dummy data cannot be optimized in any way.

After analysing the privacy of the algorithm, the authors also analyse its security and resilience by performing a Byzantine attack. The main objective of the attack is that the adversary will aim to be selected as the leader node. As the training process is a competition against the rest of the users and demands a large amount of

computational sources, the adversary might claim a fake accuracy level higher than the real performance of their model. Then, when selected as leader, this adversary performs a Gaussian attack, aggregating random values from the Gaussian distribution as model parameters, instead of aggregating the actual model parameters from the other local updates. The attack was performed with one third of the users being malicious, as this is the usual tolerance threshold for distributed systems. In the paper of the algorithm [42] it can be seen in Fig. 9 that the performance of the system is the same with and without malicious Byzantine clients, therefore proving the resilience of the algorithm for these type of attacks.

This englobes the analysis made by the authors on the algorithm, with also some insight about performance, scalability and efficiency made in the implementation evaluation section. As a conclusion the authors state that ZKPoT is a secure and scalable blockchain secure system that corrects the shortcomings of its predecessor algorithms by including FL in the training process, improving security and privacy notably.

5 Conclusion

This thesis explored federated learning from the basics, analysed the theory behind it, explained its strengths and weaknesses, and finally, studied the ZKPoT algorithm, a novel implementation related with zero knowledge cryptography. This conclusion will explain the motivation behind choosing federated learning for this dissertation, as well as the above described algorithms, and will give more insight in the practical usage of federated learning in the actual technology and possible future applications of it.

Federated learning is an innovative and established type framework that in the past years gained real traction due to its potential with AI training. Many different fields have explored the implementation of federated learning and in some cases this technique has become the state-of-art practice. Some examples of these applications are:

Google's Gboard feature is based on federated learning [133], which is used to learn words and phrases with the objective of making speech and typing predictions. The algorithm uses the typed and spoken data from the user, trains the model locally and sends the updates to Google's server. The user can even change and personalize the federated learning settings of their device [4], mainly for data privacy matters. These kind of techniques have gained popularity in the mobile and IoT sector, as it is becoming increasingly common that edge devices like smartphones or smartwatches use federated learning for personalization purposes. The devices train the model locally for features like predictive text or voice and app or advertisement recommendations. This practice enables a more personalized experience with the client, as well as constant update of the devices. Also in a more general aspect, this frameworks give companies a better understanding of how the product is being used by their clients, without disclosing their data, which is valuable for corporate level decisions.

Healthcare is also a sector with increased usage of federated learning methods. The field benefits greatly from the protection that federated learning gives to data, as it is against regulations to disclose any patient data. Therefore this data can inserted in the federated learning framework and it is possible to use it or share it in this safe way, providing great benefits to the industry or research. For example, in 2020, 20 hospitals collaborated on training a federated learning based AI model, without the exchange of raw patient data, to predict clinical outcomes of patients with COVID-19. This model called EXAM [29] (electronic medical record X-ray AI model) used data from chest CTs and X-rays to make predictions, obtaining over a 92% AUC score. Federated learning has been also used in healthcare research, with some pharmaceutical companies creating the platform MELLODDY [60]. This platform allows companies to train predictive models for drug discovery collectively, without sharing any private patient data. In this platform, every collaborator trains the model locally and then the updates are centrally aggregated, making sure corporate and privacy restrictions are preserved.

Federated learning has also become standard in the finance sector, where some banks use this type of framework to detect money laundering or fraud. The computing sandbox SWIFT [2] (Society for Worldwide Interbank Financial Telecom-

munication) is an example of this application. There are more than 11500 banks collaborating worldwide in this project, which is helping these institutions fight fraud and save substantial amounts of money. The banks train locally their model using fingerprints from payment networks, KYC data and beneficiary information, then send the update to the global model. This method established in 2024 has been a great success, resulting in a 25% increase in detection accuracy over previous traditional and individualized techniques, while also making sure privacy is preserved.

As discussed, federated learning is becoming a useful tool in the real world, used as a safe support framework for AI model training. The motivation for choosing this topic was to find a connection between cryptography and AI that would have a tangible real-world impact. Knowing that this kind of frameworks are utilised in practice was key when deciding to have federated learning as the main topic of the dissertation. Federated learning had already a track record of implementations, but it is a type of framework that is being researched and improved currently. This gave the opportunity of having sufficient reliable literature to support the theoretical basis of the thesis while also enabling to explore novel and more recent literature that would connect federated learning with other areas of cryptography. This was the main reason why the ZKPoT algorithm was chosen for this thesis, as it recent and innovative, combining the zero knowledge and blockchain cryptographic technologies in conjunction with federated learning. ZKPoT proved interesting to study as it addressed issues that similar predecessor algorithms had. For example, the frameworks zkFL [128], zkFDL [6] and ZKP-FL [132] are algorithms that combine federated learning with zero knowledge but rely on a central server for the model aggregation, which could lead to privacy leaks when information is sent to the unverified main server. ZKPoT instead relies on the consensus achievement procedure to choose a trusted leader for the aggregation. However, ZKPoT was not the first framework to integrate consensus, as the protocol ZPoL [141] used zero knowledge convolutional neural networks (zkCNN) [83] to verify the accuracy of the methods. This method, though, proved not suitable for efficient verification as several rounds of communication are needed between verifier and prover. Therefore, to address this limitation, ZKPoT proposes a different structure for more efficient and scalable verification, while also protecting data privacy against unverified parties.

Thus, this framework was chosen as it was novel and provided significant improvements compared to predecessors, while also having a clear applied purpose. It is also a tested algorithm that has proven to be robust and also scalable. This choice was also attractive in the mathematical and cryptographic side of the thesis, as it was interesting to explore applications of zero knowledge proofs and describe them in a formal way.

However, selecting such a novel algorithm has also its shortcomings as, despite its great technology, security analysis from the authors and improvements over other protocols that use similar techniques, it is still an unproven algorithm that has only been tested in academic environments without a real industrial application. This is also the case of many other cutting-edge federated learning algorithms, as research usually evolves faster than applied need.In fact, the previously mentioned predecessor algorithms, like zkFL, zkFDL, ZKP-FL or ZPoL and also ZKPoT have

only been implemented in research settings and not in the real world, which leaves questions on how they will perform in a more scalable and less ideal environment.

There are some reasons as why federated learning frameworks based on zero knowledge proofs are not yet implemented, such as: ZKPs are computationally heavy, sometimes taking up to minutes to generate them per round of training. Federated learning models are also not the perfect match for ZKPs, as neural networks usually work with non-linear functions, whereas ZKPs work with linear arithmetic circuits over finite fields, therefore it is necessary to transform and approximate functions, losing information in the process. Also, ZKPs are hard to implement as there are not standardized tools for some types of circuits, specially when trying to make it compatible with ML algorithms. Finally, from the industry point of view, there is a lack of incentive on developing such secure frameworks, as usually simpler FL security methods like secure aggregation or differential privacy are secure enough for the regulations they need to comply with. Therefore, including ZKPs is seen as overcomplicated and only would be useful in situations where trust is extremely minimal.

In conclusion, ZK-FL is a powerful and extremely secure tool that has great potential to develop distributed model training, however, before it becomes a reliable practice we might need to see improvements in ZKP computational efficiency, better standardised implementation tools and more interest and demand for verifiable privacy in real world applications. However, with the rise AI and ML models, FL has gotten a great deal of interest and has already been applied in many fields. Therefore, it is only natural that research in this topic will advance and soon FL will be even more present in devices and industry applied technology in the future.

References

[1] 1973. URL: https://rodsmith.nz/wp-content/uploads/Lighthill_1973_Report.pdf.

[2] Oct. 2024. URL: https://www.swift.com/es/node/309839.

[3] URL: https://www.ibm.com/history/deep-blue.

[4] URL: https://support.google.com/gboard/answer/12373137?hl=en#zippy=%2Cfederated-learning.

[5] Mojtaba Ahmadi and Reza Nourmohammadi. *zkDFL: An efficient and privacy-preserving decentralized federated learning with zero-knowledge proof*. 2024. arXiv: 2312.04579 [cs.CR]. URL: https://arxiv.org/abs/2312.04579.

[6] Mojtaba Ahmadi and Reza Nourmohammadi. "zkFDL: An efficient and privacy-preserving decentralized federated learning with zero knowledge proof". In: Feb. 2024, pp. 1–10. DOI: 10.1109/ICAIC60265.2024.10433831.

[7] Galen Andrew et al. *Differentially Private Learning with Adaptive Clipping*. 2022. arXiv: 1905.03871 [cs.LG]. URL: https://arxiv.org/abs/1905.03871.

[8] Eugene Bagdasaryan et al. "How To Backdoor Federated Learning". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 2938–2948. URL: https://proceedings.mlr.press/v108/bagdasaryan20a.html.

[9] Juan Benet. *IPFS - Content Addressed, Versioned, P2P File System*. 2014. arXiv: 1407.3561 [cs.NI]. URL: https://arxiv.org/abs/1407.3561.

[10] Maurizio Binello. *My quest for knowledge about "Zero-knowledge"*. 2019. URL: https://www.zeroknowledgeblog.com/.

[11] Nir Bitansky et al. *The Hunting of the SNARK*. Cryptology ePrint Archive, Paper 2014/580. 2014. URL: https://eprint.iacr.org/2014/580.

[12] Peva Blanchard et al. "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf.

[13] Keith Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1175–1191. ISBN: 9781450349468. DOI: 10.1145/3133956.3133982. URL: https://doi.org/10.1145/3133956.3133982.

[14] Steven Borowiec. *AlphaGo Seals 4-1 victory over go grandmaster Lee Sedol*. Mar. 2016. URL: https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol.

[15] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[16] Di Cao et al. "Understanding Distributed Poisoning Attack in Federated Learning". In: *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. 2019, pp. 233–239. DOI: 10.1109/ICPADS47876.2019.00042.

[17] Xiaoyu Cao et al. *FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping*. 2022. arXiv: 2012.13995 [cs.CR]. URL: https://arxiv.org/abs/2012.13995.

[18] Di Chai et al. "Secure Federated Matrix Factorization". In: *IEEE Intelligent Systems* 36.5 (Sept. 2021), pp. 11–20. ISSN: 1941-1294. DOI: 10.1109/mis.2020.3014880. URL: http://dx.doi.org/10.1109/MIS.2020.3014880.

[19] Jianhua Chen et al. *Hasse theorem – an elementary approach*. 2012. arXiv: 1212.2535 [math.GM]. URL: https://arxiv.org/abs/1212.2535.

[20] Jingxue Chen et al. "UFL: Unlinkable Federated Learning Through Shuffle and Shamir's Secret Sharing". In: *Advanced Data Mining and Applications*. Ed. by Quan Z. Sheng et al. Singapore: Springer Nature Singapore, 2025, pp. 239–253.

[21] Lvjun Chen et al. "Secure and efficient federated learning via novel multi-party computation and compressed sensing". In: *Information Sciences* 667 (2024), p. 120481. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2024.120481. URL: https://www.sciencedirect.com/science/article/pii/S0020025524003943.

[22] Yao Chen et al. "Federated Learning Attacks and Defenses: A Survey". In: *2022 IEEE International Conference on Big Data (Big Data)*. 2022, pp. 4256–4265. DOI: 10.1109/BigData55660.2022.10020431.

[23] Yao Chen et al. *Federated Learning Attacks and Defenses: A Survey*. 2022. arXiv: 2211.14952 [cs.CR]. URL: https://arxiv.org/abs/2211.14952.

[24] Yu Chen et al. "A training-integrity privacy-preserving federated learning scheme with trusted execution environment". In: *Information Sciences* 522 (2020), pp. 69–79. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2020.02.037. URL: https://www.sciencedirect.com/science/article/pii/S0020025520301201.

[25] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. DOI: 10.1007/bf00994018.

[26] Gabriela F. Cretu et al. "Casting out Demons: Sanitizing Training Data for Anomaly Sensors". In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. 2008, pp. 81–95. DOI: 10.1109/SP.2008.11.

[27] Yanbo Dai and Songze Li. *Chameleon: Adapting to Peer Images for Planting Durable Backdoors in Federated Learning*. 2023. arXiv: 2304.12961 [cs.LG]. URL: https://arxiv.org/abs/2304.12961.

[28] Deepesh Data, Linqi Song, and Suhas Diggavi. *Data Encoding for Byzantine-Resilient Distributed Optimization*. 2020. arXiv: 1907.02664 [cs.DC]. URL: https://arxiv.org/abs/1907.02664.

[29] Ittai Dayan et al. "Federated learning for predicting clinical outcomes in patients with COVID-19". In: *Nature Medicine* 27.10 (Sept. 2021), pp. 1735–1743. DOI: 10.1038/s41591-021-01506-3.

[30] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[31] Desmos. *Desmos Graphing Calculator*. https://www.desmos.com/calculator. Accessed: July 28, 2025. 2011.

[32] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[33] John R. Douceur. "The sybil attack". In: *Lecture Notes in Computer Science* (2002), pp. 251–260. DOI: 10.1007/3-540-45748-8_24.

[34] Cynthia Dwork and Aaron Roth. "The algorithmic foundations of Differential Privacy". In: *Foundations and Trends R in Theoretical Computer Science* 9 (2014). DOI: 10.1561/9781601988195.

[35] Sannara EK et al. "A Federated Learning Aggregation Algorithm for Pervasive Computing: Evaluation and Comparison". In: *2021 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, Mar. 2021, pp. 1–10. DOI: 10.1109/percom50583.2021.9439129. URL: http://dx.doi.org/10.1109/PERCOM50583.2021.9439129.

[36] Chen Fang et al. "Privacy-preserving and communication-efficient federated learning in Internet of Things". In: *Computers Security* 103 (2021), p. 102199. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2021.102199.

[37] FedAI. *Fedai Home*. Apr. 2020. URL: https://www.fedai.org/.

[38] FedAI. *Utilization of fate in risk management of credit in small and micro enterprises*. Jan. 2020. URL: https://www.fedai.org/cases/utilization-of-fate-in-risk-management-of-credit-in-small-and-micro-enterprises/.

[39] Boyuan Feng et al. *ZEN: An Optimizing Compiler for Verifiable, Zero Knowledge Neural Network Inferences*. Cryptology ePrint Archive, Paper 2021/087. 2021. URL: https://eprint.iacr.org/2021/087.

[40] Yunhao Feng et al. "A survey of security threats in Federated Learning". In: *Complex amp; Intelligent Systems* 11.2 (Jan. 2025). DOI: 10.1007/s40747-024-01664-0.

[41] Yann Fraboni, Richard Vidal, and Marco Lorenzi. "Free-rider Attacks on Model Aggregation in Federated Learning". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by Arindam Banerjee and Kenji Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 13–15 Apr 2021, pp. 1846–1854. URL: https://proceedings.mlr.press/v130/fraboni21a.html.

[42] Tianxing Fu et al. *Zero-Knowledge Proof-Based Consensus for Blockchain-Secured Federated Learning*. 2025. arXiv: 2503.13255 [cs.DC]. URL: https://arxiv.org/abs/2503.13255.

[43] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Paper 2019/953. 2019. URL: https://eprint.iacr.org/2019/953.

[44] Mark Gales and Steve Young. "The application of Hidden Markov models in speech recognition". In: *Foundations and Trends® in Signal Processing* 1.3 (2007), pp. 195–304. DOI: 10.1561/2000000004.

[45] Roberto Garcia Alvira and Miguel Ángel Marco Buzunárriz. "Zero Knowledge Proofs". In: *zaguan.unizar* (2023), pp. 1–25.

[46] GeeksforGeeks. *Types of federated learning in Machine Learning*. May 2024. URL: https://www.geeksforgeeks.org/types-of-federated-learning-in-machine-learning/.

[47] Rosario Gennaro et al. "Quadratic Span Programs and Succinct NIZKs without PCPs". In: vol. 7881. May 2013. ISBN: 978-3-642-38347-2. DOI: 10.1007/978-3-642-38348-9_37.

[48] Robin C. Geyer, Tassilo Klein, and Moin Nabi. *Differentially Private Federated Learning: A Client Level Perspective*. 2018. arXiv: 1712.07557 [cs.CR]. URL: https://arxiv.org/abs/1712.07557.

[49] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. DOI: 10.1137/0218012. eprint: https://doi.org/10.1137/0218012. URL: https://doi.org/10.1137/0218012.

[50] Andrew Gooday. *Understanding the types of Federated Learning*. Dec. 2024. URL: https://openmined.org/blog/federated-learning-types/.

[51] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: https://arxiv.org/abs/1406.2661.

[52] Jens Groth. "On the size of pairing-based non-interactive arguments". In: *Lecture Notes in Computer Science* (2016), pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11.

[53] Manisha Guduri et al. "Blockchain-Based Federated Learning Technique for Privacy Preservation and Security of Smart Electronic Health Records". In: *IEEE Transactions on Consumer Electronics* 70.1 (2024), pp. 2608–2617. DOI: 10.1109/TCE.2023.3315415.

[54] Wei Guo et al. "A comprehensive survey of Federated Transfer Learning: Challenges, methods and applications". In: *Frontiers of Computer Science* 18.6 (July 2024). DOI: 10.1007/s11704-024-40065-x.

[55] Florian Hartmann. *Florian/Federated-Learning*. 2018. URL: https://github.com/florian/federated-learning.

[56] Florian Hartmann. *Florian/Federated-Learning-Addon*. 2018. URL: https://github.com/florian/federated-learning-addon.

[57] Florian Hartmann and Wolfgang Mulzer. "Federated Learning". PhD thesis. Freie Universität Berlin, 2018, pp. 1–78.

[58] Ali Hatamizadeh et al. *GradViT: Gradient Inversion of Vision Transformers*. 2022. arXiv: 2203.11894 [cs.CV]. URL: https://arxiv.org/abs/2203.11894.

[59] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[60] Wouter Heyndrickx et al. "Melloddy: Cross-pharma Federated Learning at unprecedented scale unlocks benefits in QSAR without compromising proprietary information". In: *Journal of Chemical Information and Modeling* 64.7 (Aug. 2023), pp. 2331–2344. DOI: 10.1021/acs.jcim.3c00799.

[61] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural Computation* 18.7 (July 2006), pp. 1527–1554. DOI: 10.1162/neco.2006.18.7.1527.

[62] Kai Hu et al. "An overview of implementing security and privacy in federated learning". In: *Artificial Intelligence Review* 57 (July 2024). DOI: 10.1007/s10462-024-10846-8.

[63] Ting Wei Huang and Christopher Smith. "The History of Artificial Intelligence". In: 2006. URL: https://api.semanticscholar.org/CorpusID:263599500.

[64] Ibm. *The history of Artificial Intelligence*. Mar. 2025. URL: https://www.ibm.com/think/topics/history-of-artificial-intelligence.

[65] Yuang Jiang et al. *Model Pruning Enables Efficient Federated Learning on Edge Devices*. 2022. arXiv: 1909.12326 [cs.LG]. URL: https://arxiv.org/abs/1909.12326.

[66] John Jumper et al. "Highly accurate protein structure prediction with alphafold". In: *Nature* 596.7873 (July 2021), pp. 583–589. DOI: 10.1038/s41586-021-03819-2.

[67] Sameera K.M. et al. "Privacy-preserving in Blockchain-based Federated Learning systems". In: *Computer Communications* 222 (June 2024), pp. 38–67. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2024.04.024. URL: http://dx.doi.org/10.1016/j.comcom.2024.04.024.

[68] Peter Kairouz et al. *Advances and Open Problems in Federated Learning*. 2021. arXiv: 1912.04977 [cs.LG]. URL: https://arxiv.org/abs/1912.04977.

[69] Youssef Khazbak, Tianxiang Tan, and Guohong Cao. "MLGuard: Mitigating Poisoning Attacks in Privacy Preserving Distributed Collaborative Learning". In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 2020, pp. 1–9. DOI: 10.1109/ICCCN49398.2020.9209670.

[70] Tung Kieu et al. "Outlier Detection for Time Series with Recurrent Autoencoder Ensembles". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 2725–2732. DOI: 10.24963/ijcai.2019/378. URL: https://doi.org/10.24963/ijcai.2019/378.

[71] Pang Wei Koh, Jacob Steinhardt, and Percy Liang. *Stronger Data Poisoning Attacks Break Data Sanitization Defenses*. 2021. arXiv: 1811.00741 [stat.ML]. URL: https://arxiv.org/abs/1811.00741.

[72] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[73] K Naveen Kumar, Reshmi Mitra, and C Krishna Mohan. "Revamping Federated Learning Security from a Defender's Perspective: A Unified Defense with Homomorphic Encrypted Data Space". In: *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024, pp. 24387–24397. DOI: 10.1109/CVPR52733.2024.02302.

[74] Anusha Lalitha et al. "Fully Decentralized Federated Learning". In: *Bayesian deep learning* 140 (2018).

[75] Haowei Li et al. *FedFQ: Federated Learning with Fine-Grained Quantization*. 2024. arXiv: 2408.08977 [cs.DC]. URL: https://arxiv.org/abs/2408.08977.

[76] Jichang Li et al. *FedDiv: Collaborative Noise Filtering for Federated Learning with Noisy Labels*. 2024. arXiv: 2312.12263 [cs.CV]. URL: https://arxiv.org/abs/2312.12263.

[77] Jie Li et al. "Improve individual fairness in federated learning via adversarial training". In: *Computers Security* 132 (2023), p. 103336. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2023.103336. URL: https://www.sciencedirect.com/science/article/pii/S0167404823002468.

[78] Liping Li et al. *RSA: Byzantine-Robust Stochastic Aggregation Methods for Distributed Learning from Heterogeneous Datasets*. 2019. arXiv: 1811.03761 [cs.LG]. URL: https://arxiv.org/abs/1811.03761.

[79] Lu Li, Jiwei Qin, and Jintao Luo. "A Blockchain-Based Federated-Learning Framework for Defense against Backdoor Attacks". In: *Electronics* 12 (June 2023), p. 2500. DOI: 10.3390/electronics12112500.

[80] Yanli Li et al. "Enhancing federated learning robustness in adversarial environment through clustering Non-IID features". In: *Computers Security* 132 (2023), p. 103319. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2023.103319. URL: https://www.sciencedirect.com/science/article/pii/S0167404823002298.

[81] Yanli Li et al. *Threats and Defenses in Federated Learning Life Cycle: A Comprehensive Survey and Challenges*. 2024. arXiv: 2407.06754 [cs.DC]. URL: https://arxiv.org/abs/2407.06754.

[82] Zhuohang Li et al. *Auditing Privacy Defenses in Federated Learning via Generative Gradient Leakage*. 2022. arXiv: 2203.15696 [cs.CV]. URL: https://arxiv.org/abs/2203.15696.

[83] Tianyi Liu, Xiang Xie, and Yupeng Zhang. "zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 2968–2985. ISBN: 9781450384544. DOI: 10.1145/3460120.3485379. URL: https://doi.org/10.1145/3460120.3485379.

[84] Yunlong Lu et al. "Blockchain and Federated Learning for Privacy-Preserved Data Sharing in Industrial IoT". In: *IEEE Transactions on Industrial Informatics* PP (Sept. 2019), pp. 1–1. DOI: 10.1109/TII.2019.2942190.

[85] John McCarthy et al. *A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE*. Aug. 1955. URL: https://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html.

[86] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. DOI: 10.1007/bf02478259.

[87] Brendan McMahan and Daniel Ramage. *Federated learning: Collaborative machine learning without centralized training*. Apr. 2017. URL: https://research.google/blog/federated-learning-collaborative-machine-learning-without-centralized-training-data/.

[88] H. Brendan McMahan et al. *Communication-Efficient Learning of Deep Networks from Decentralized Data*. 2023. arXiv: 1602.05629 [cs.LG]. URL: https://arxiv.org/abs/1602.05629.

[89] H. Brendan McMahan et al. *Learning Differentially Private Recurrent Language Models*. 2018. arXiv: 1710.06963 [cs.LG]. URL: https://arxiv.org/abs/1710.06963.

[90] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. *The Hidden Vulnerability of Distributed Learning in Byzantium*. 2018. arXiv: 1802.07927 [stat.ML]. URL: https://arxiv.org/abs/1802.07927.

[91] Victor Miller. "The Weil Pairing, and Its Efficient Calculation". In: *J. Cryptology* 17 (Sept. 2004), pp. 235–261. DOI: 10.1007/s00145-004-0315-8.

[92] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[93] Fan Mo et al. *PPFL: Privacy-preserving Federated Learning with Trusted Execution Environments*. 2021. arXiv: 2104.14380 [cs.CR]. URL: https://arxiv.org/abs/2104.14380.

[94] Viraaji Mothukuri et al. "A survey on security and privacy of federated learning". In: *Future Gener. Comput. Syst.* 115.C (Feb. 2021), pp. 619–640. ISSN: 0167-739X. DOI: 10.1016/j.future.2020.10.007. URL: https://doi.org/10.1016/j.future.2020.10.007.

[95] Xianyu Mu et al. "FLAP: Federated Learning Aggregation Scheme Based on Privileged Secret Sharing". In: *2023 International Conference on Networking and Network Applications (NaNA)*. 2023, pp. 588–594. DOI: 10.1109/NaNA60121.2023.00102.

[96] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. *Local and Central Differential Privacy for Robustness and Privacy in Federated Learning*. 2022. arXiv: 2009.03561 [cs.CR]. URL: https://arxiv.org/abs/2009.03561.

[97] Alex Ozdemir and Dan Boneh. *Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets*. Cryptology ePrint Archive, Paper 2021/1530. 2021. URL: https://eprint.iacr.org/2021/1530.

[98] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology — EUROCRYPT '99*. Ed. by Jacques Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[99] Jungwuk Park et al. "Sageflow: Robust Federated Learning against Both Stragglers and Adversaries". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 840–851. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/076a8133735eb5d7552dc195b125a454-Paper.pdf.

[100] Bryan Parno et al. "Pinocchio: Nearly Practical Verifiable Computation". In: vol. 59. May 2013, pp. 238–252. DOI: 10.1109/SP.2013.47.

[101] Youyang Qu et al. "A Blockchained Federated Learning Framework for Cognitive Computing in Industry 4.0 Networks". In: *IEEE Transactions on Industrial Informatics* 17.4 (2021), pp. 2964–2973. DOI: 10.1109/TII.2020.3007817.

[102] Amos R. Omondi. "Elliptic-Curve Basics". In: *Cryptography Arithmetic: Algorithms and Hardware Architectures*. Cham: Springer International Publishing, 2020, pp. 225–241. ISBN: 978-3-030-34142-8. DOI: 10.1007/978-3-030-34142-8_8. URL: https://doi.org/10.1007/978-3-030-34142-8_8.

[103] Paritosh Ramanan and Kiyoshi Nakayama. *BAFFLE : Blockchain Based Aggregator Free Federated Learning*. 2020. arXiv: 1909.07452 [cs.LG]. URL: https://arxiv.org/abs/1909.07452.

[104] Aditya Ramesh et al. *Zero-Shot Text-to-Image Generation*. 2021. arXiv: 2102.12092 [cs.CV]. URL: https://arxiv.org/abs/2102.12092.

[105] Ishai Rosenberg et al. "Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain". In: *ACM Comput. Surv.* 54.5 (May 2021). ISSN: 0360-0300. DOI: 10.1145/3453158. URL: https://doi.org/10.1145/3453158.

[106] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519. URL: http://dx.doi.org/10.1037/h0042519.

[107] Andrew Ross. *Federated-learning/SRC/server/readme.md at master · pair-code/federated-learning*. 2018. URL: https://github.com/PAIR-code/federated-learning/blob/master/src/server/README.md.

[108] Arthur L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 3.3 (July 1959), pp. 210–229.

[109] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. *Clustered Federated Learning: Model-Agnostic Distributed Multi-Task Optimization under Privacy Constraints*. 2019. arXiv: 1910.01991 [cs.LG]. URL: https://arxiv.org/abs/1910.01991.

[110] Felix Sattler et al. "On the Byzantine Robustness of Clustered Federated Learning". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 8861–8865. DOI: 10.1109/ICASSP40776.2020.9054676.

[111] Adi Shamir. "How to share a secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[112] G. L. Shaw. "Donald Hebb: The Organization of Behavior". In: *Brain Theory* (1949), pp. 231–233. DOI: 10.1007/978-3-642-70911-1_15.

[113] Shiqi Shen, Shruti Tople, and Prateek Saxena. "Auror: defending against poisoning attacks in collaborative deep learning systems". In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACSAC '16. Los Angeles, California, USA: Association for Computing Machinery, 2016, pp. 508–519. ISBN: 9781450347716. DOI: 10.1145/2991079.2991125. URL: https://doi.org/10.1145/2991079.2991125.

[114] Ghazaleh Shirvani, Saeid Ghasemshirazi, and Behzad Beigzadeh. *Federated Learning: Attacks, Defenses, Opportunities, and Challenges*. 2024. arXiv: 2403.06067 [cs.CR]. URL: https://arxiv.org/abs/2403.06067.

[115] R.J. Solomonoff. "A formal theory of inductive inference. Part II". In: *Information and Control* 7.2 (1964), pp. 224–254. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(64)90131-7. URL: https://www.sciencedirect.com/science/article/pii/S0019995864901317.

[116] Kangkang Sun et al. "Joint Top-K Sparsification and Shuffle Model for Communication Privacy Accuracy Tradeoffs in Federated-Learning-Based IoV". In: *IEEE Internet of Things Journal* 11.11 (2024), pp. 19721–19735. DOI: 10.1109/JIOT.2024.3370991.

[117] Rahim Taheri et al. "Fed-IIoT: A Robust Federated Malware Detection Architecture in Industrial IoT". In: *IEEE Transactions on Industrial Informatics* 17.12 (2021), pp. 8442–8452. DOI: 10.1109/TII.2020.3043458.

[118] team-rareskills 6 months ago team-rareskills and team-rareskills 7 months ago team-rareskills. *Elliptic curve point addition*. Nov. 2024. URL: https://www.rareskills.io/post/elliptic-curve-addition.

[119] Florian Tramèr et al. *Ensemble Adversarial Training: Attacks and Defenses*. 2020. arXiv: 1705.07204 [stat.ML]. URL: https://arxiv.org/abs/1705.07204.

[120] John Tromp and Gunnar Farnebäck. "Combinatorics of Go". In: *Computers and Games*. Ed. by H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. Donkers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 84–99.

[121] Stacey Truex et al. *LDP-Fed: Federated Learning with Local Differential Privacy*. 2020. arXiv: 2006.03637 [cs.LG]. URL: https://arxiv.org/abs/2006.03637.

[122] A. M. Turing. "Computing Machinery and Intelligence". English. In: *Mind*. New Series 59.236 (1950), pp. 433–460. ISSN: 00264423. URL: http://www.jstor.org/stable/2251299.

[123] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[124] Riad Wahby et al. "Doubly-Efficient zkSNARKs Without Trusted Setup". In: May 2018, pp. 926–943. DOI: 10.1109/SP.2018.00060.

[125] Wei Wan et al. "Shielding Federated Learning: A New Attack Approach and Its Defense". In: *IEEE Wireless Communications and Networking Conference, WCNC 2021, Nanjing, China, March 29 - April 1, 2021*. IEEE, 2021, pp. 1–7. DOI: 10.1109/WCNC49053.2021.9417334. URL: https://doi.org/10.1109/WCNC49053.2021.9417334.

[126] Chang Wang et al. *Hybrid Differentially Private Federated Learning on Vertically Partitioned Data*. 2020. arXiv: 2009.02763 [cs.LG]. URL: https://arxiv.org/abs/2009.02763.

[127] Zhipeng Wang et al. *zkFL: Zero-Knowledge Proof-based Gradient Aggregation for Federated Learning*. 2024. arXiv: 2310.02554 [cs.AI]. URL: https://arxiv.org/abs/2310.02554.

[128] Zhipeng Wang et al. *zkFL: Zero-Knowledge Proof-based Gradient Aggregation for Federated Learning*. 2025. arXiv: 2310.02554 [cs.AI]. URL: https://arxiv.org/abs/2310.02554.

[129] Webank. *Webank: Home page*. URL: https://www.webank.it/webankpub/wbresp/home.do.

[130] Joseph Weizenbaum. "ELIZA—a computer program for the study of natural language communication between man and machine". In: *Commun. ACM* 9.1 (Jan. 1966), pp. 36–45. ISSN: 0001-0782. DOI: 10.1145/365153.365168. URL: https://doi.org/10.1145/365153.365168.

[131] Yueqi Xie et al. *Robust Federated Learning against both Data Heterogeneity and Poisoning Attack via Aggregation Optimization*. 2022. arXiv: 2211.05554 [cs.LG]. URL: https://arxiv.org/abs/2211.05554.

[132] Zhibo Xing et al. *Zero-Knowledge Proof-based Practical Federated Learning on Blockchain*. 2023. arXiv: 2304.05590 [cs.CR]. URL: https://arxiv.org/abs/2304.05590.

[133] Zheng Xu et al. *Federated Learning of Gboard Language Models with Differential Privacy*. 2023. arXiv: 2305.18465 [cs.LG]. URL: https://arxiv.org/abs/2305.18465.

[134] Qiang Yang et al. *Federated Machine Learning: Concept and Applications*. 2019. arXiv: 1902.04885 [cs.AI]. URL: https://arxiv.org/abs/1902.04885.

[135] Andrew C. Yao. "Protocols for secure computations". In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.

[136] Abbas Yazdinejad et al. "Block Hunter: Federated Learning for Cyber Threat Hunting in Blockchain-Based IIoT Networks". In: *IEEE Transactions on Industrial Informatics* 18.11 (2022), pp. 8356–8366. DOI: 10.1109/TII.2022.3168011.

[137] Gokul Yenduri et al. *Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions*. 2023. arXiv: 2305.10435 [cs.CL]. URL: https://arxiv.org/abs/2305.10435.

[138] Dong Yin et al. *Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates*. 2021. arXiv: 1803.01498 [cs.LG]. URL: https://arxiv.org/abs/1803.01498.

[139] Hongxu Yin et al. *See through Gradients: Image Batch Recovery via GradInversion*. 2021. arXiv: 2104.07586 [cs.LG]. URL: https://arxiv.org/abs/2104.07586.

[140] Haobo Zhang et al. *Understanding Deep Gradient Leakage via Inversion Influence Functions*. 2024. arXiv: 2309.13016 [cs.LG]. URL: https://arxiv.org/abs/2309.13016.

[141] Heyi Zhang et al. "Integrating Blockchain and Deep Learning into Extremely Resource-Constrained IoT: An Energy-Saving Zero-Knowledge PoL Approach". English. In: *IEEE Internet of Things Journal* 11.3 (Feb. 2024). Publisher Copyright: © 2014 IEEE., pp. 3881–3895. ISSN: 2327-4662. DOI: 10.1109/JIOT.2023.3280069.

[142] Yang Zhao et al. *Local Differential Privacy based Federated Learning for Internet of Things*. 2020. arXiv: 2004.08856 [cs.CR]. URL: https://arxiv.org/abs/2004.08856.

[143] Xiaokang Zhou et al. "2D Federated Learning for Personalized Human Activity Recognition in Cyber-Physical-Social Systems". In: *IEEE Transactions on Network Science and Engineering* 9.6 (2022), pp. 3934–3944. DOI: 10.1109/TNSE.2022.3144699.

[144] Ligeng Zhu, Zhijian Liu, and Song Han. *Deep Leakage from Gradients*. 2019. arXiv: 1906.08935 [cs.LG]. URL: https://arxiv.org/abs/1906.08935.