Robust and privacy preserving distributed machine learning

Rania Talbi

THESE DE DOCTORAT DE L'UNIVERSITE DE LYON

opérée au sein de

l'INSA de Lyon

Ecole Doctorale **N° 512**

Mathématiques et Informatique (InfoMaths)

Spécialité/discipline de doctorat : INFORMATIQUE

NNT 2021LYSEI077

Robust and Privacy Preserving Distributed Machine Learning

Soutenue publiquement le 19/11/2021, par :

Rania Talbi

Devant le jury composé de :

Benjamin Nguyen	Professeur des Universités, INSA Val de Loire	Rapporteur
Marc Tommasi	Professeur des Universités, Université de Lille	Rapporteur
Lydia Chen	Maître de conférences, TU Delft	Examinatrice
Lionel Brunie	Professeur des Universités, INSA-Lyon	Examinateur
Bouchenak Sara	Professeur des Universités, INSA-Lyon	Directrice de thèse

Département FEDORA – INSA Lyon - Ecoles Doctorales

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	CHIMIE DE LYON https://www.edchimie-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage secretariat@edchimie-lyon.fr	M. Stéphane DANIELE C2P2-CPE LYON-UMR 5265 Bâtiment F308, BP 2077 43 Boulevard du 11 novembre 1918 69616 Villeurbanne directeur@edchimie-lyon.fr
E.E.A.	ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE https://edeea.universite-lyon.fr Sec. : Stéphanie CAUVIN Bâtiment Direction INSA Lyon Tél : 04.72.43.71.70 secretariat.edeea@insa-lyon.fr	M. Philippe DELACHARTRE INSA LYON Laboratoire CREATIS Bâtiment Blaise Pascal, 7 avenue Jean Capelle 69621 Villeurbanne CEDEX Tél : 04.72.43.88.63 philippe.delachartre@insa-lyon.fr
E2M2	ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION http://e2m2.universite-lyon.fr Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.e2m2@univ-lyon1.fr	M. Philippe NORMAND Université Claude Bernard Lyon 1 UMR 5557 Lab. d'Ecologie Microbienne Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69 622 Villeurbanne CEDEX philippe.normand@univ-lyon1.fr
EDISS	INTERDISCIPLINAIRE SCIENCES-SANTÉ http://ediss.universite-lyon.fr Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.ediss@univ-lyon1.fr	Mme Sylvie RICARD-BLUM Institut de Chimie et Biochimie Moléculaires et Supramoléculaires (ICBMS) - UMR 5246 CNRS - Université Lyon 1 Bâtiment Raulin - 2ème étage Nord 43 Boulevard du 11 novembre 1918 69622 Villeurbanne Cedex Tél : +33(0)4 72 44 82 32 sylvie.ricard-blum@univ-lyon1.fr
INFOMATHS	INFORMATIQUE ET MATHÉMATIQUES http://edinfomaths.universite-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 infomaths@univ-lyon1.fr	M. Hamamache KHEDDOUCI Université Claude Bernard Lyon 1 Bât. Nautibus 43, Boulevard du 11 novembre 1918 69 622 Villeurbanne Cedex France Tél : 04.72.44.83.69 hamamache.kheddouci@univ-lyon1.fr
Matériaux	MATÉRIAUX DE LYON http://ed34.universite-lyon.fr Sec. : Yann DE ORDENANA Tél : 04.72.18.62.44 yann.de-ordenana@ec-lyon.fr	M. Stéphane BENAYOUN Ecole Centrale de Lyon Laboratoire LTDS 36 avenue Guy de Collongue 69134 Ecully CEDEX Tél : 04.72.18.64.37 stephane.benayoun@ec-lyon.fr
MEGA	MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE http://edmega.universite-lyon.fr Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bâtiment Direction INSA Lyon mega@insa-lyon.fr	M. Jocelyn BONJOUR INSA Lyon Laboratoire CETHIL Bâtiment Sadi-Carnot 9, rue de la Physique 69621 Villeurbanne CEDEX jocelyn.bonjour@insa-lyon.fr
ScSo	ScSo* https://edsciencessociales.universite-lyon.fr Sec. : Mélina FAVETON INSA : J.Y. TOUSSAINT Tél : 04.78.69.77.79 melina.faveton@univ-lyon2.fr	M. Christian MONTES Université Lumière Lyon 2 86 Rue Pasteur 69365 Lyon CEDEX 07 christian.montes@univ-lyon2.fr

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Abstract

With the pervasiveness of digital services, huge amounts of data are nowadays continuously generated and collected. Machine Learning (ML) algorithms allow the extraction of hidden yet valuable knowledge from these data and have been applied in numerous domains, such as health care assistance, transportation, user behavior prediction, and many others. In many of these applications, data is collected from different sources and distributed training is required to learn global models over them. However, in the case of sensitive data, using traditional ML algorithms over them can lead to serious privacy breaches by leaking sensitive information about data owners.

In this thesis, we propose mechanisms allowing to enhance privacy preservation and robustness in the domain of distributed machine learning.

The first contribution of this thesis falls in the category of cryptography-based Privacy Preserving Machine Learning (PPML). Many state-of-the-art works propose cryptography-based solutions to ensure privacy preservation in distributed machine learning. Nevertheless, these works are known to induce huge overheads time and space-wise. In this line of work, we propose *PrivML* an outsourced Homomorphic Encryption-based Privacy Preserving Collaborative Machine Learning framework, that allows optimizing runtime and bandwidth consumption for widely used ML algorithms, using many techniques such as fast algorithms for large integer arithmetic, ciphertext packing, approximate computations, and parallel computing.

The other contributions of this thesis addresses the robustness issues in the domain of Federated Learning (FL). Indeed federated learning is the first framework to ensure privacy by design for distributed machine learning. Nonetheless, it has been shown that this framework is still vulnerable to many attacks, among them we find poisoning attacks, where participants deliberately use faulty training data to provoke misclassification at inference time. We demonstrate that state-of-the-art poisoning mitigation mechanisms fail to detect some poisoning attacks and propose *ARMOR*, a poisoning mitigation mechanism for Federated Learning that successfully detects these attacks, without hurting models' utility.

Keywords: Privacy, Homomorphic Encryption, Distributed Machine Learning, Robustness, Federated Learning, Poisoning Attacks.

Résumé

Avec l'omniprésence des services numériques, d'énormes quantités de données sont continuellement générées et collectées. Les algorithmes d'apprentissage automatique (ML) permettant d'extraire des connaissances précieuses à partir de ces données et ont été appliqués dans de nombreux domaines, tels que l'assistance médicale, le transport, la prédiction du comportement des utilisateurs, et bien d'autres.

Dans beaucoup de ces applications, les données sont collectées à partir de différentes sources et un entraînement distribué est nécessaire pour apprendre des modèles globaux sur ces données. Néanmoins, dans le cas de données sensibles, l'exécution d'algorithmes ML traditionnels sur ces données peut conduire à de graves violations de la vie privée en divulguant des informations sensibles sur les propriétaires des données.

Dans cette thèse, nous proposons des mécanismes permettant d'améliorer la préservation de la vie privée et la robustesse dans le domaine de l'apprentissage automatique distribué.

La première contribution de cette thèse s'inscrit dans la catégorie d'apprentissage automatique respectueux de la vie privée (PPML) basé sur la cryptographie.

De nombreux travaux de l'état de l'art proposent des solutions basées sur la cryptographie pour assurer la préservation de la vie privée dans l'apprentissage automatique distribué. Néanmoins, ces travaux sont connus par leurs énormes coûts en termes de temps d'exécution et d'espace. Dans cette lignée de travaux, nous proposons *PrivML*, un framework externalisé d'apprentissage collaboratif basé sur le chiffrement homomorphe, qui permet d'optimiser le temps d'exécution et la consommation de bande passante pour les algorithmes ML les plus utilisés, moyennant de nombreuses techniques telles que le packing, l'usage d'algorithmes optimisés de l'arithmétique multiprécision, les calculs approximatifs et le calcul parallèle.

Les autres contributions de cette thèse abordent les questions de robustesse dans le domaine de l'apprentissage fédéré.

En effet, l'apprentissage fédéré est le premier framework à garantir la préservation de la vie privée par conception dans le cadre de l'apprentissage automatique distribué. Néanmoins, il a été démontré que ce framework est toujours vulnérable à de nombreuses attaques, parmi lesquelles nous trouvons les attaques par empoisonnement, où les participants utilisent délibérément des données d'entraînement erronées pour provoquer une mauvaise classification au moment de l'inférence.

Nous démontrons que les mécanismes de mitigation de l'empoisonnement de l'état de l'art ne parviennent pas à détecter certaines attaques par empoisonnement et nous proposons *ARMOR*, un mécanisme de mitigation de l'empoisonnement pour l'apprentissage fédéré qui parvient à détecter ces attaques sans nuire à l'utilité des modèles.

Mots-clés: Préservation de la vie privée, Apprentissage automatique distribué, Chiffrement homomorphe, Robustesse, Apprentissage fédéré, Attaques par empoisonnement.

Contents

List of Acronyms

ACGAN	Auxiliary Classifier Generative Adversarial Network
DDBM	Data-Driven Business Model
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DO	Data Owner
DP	Data Provider
FedAvg	Federated Averaging
FHE	Fully Homomorphic Encryption
FL	Federated Learning
GAN	Generative Adversarial Network
GC	Garbled Circuits
HE	Homomorphic Encryption
ML	Machine Learning
MLaaS	Machine Learning as a Service
MLSP	Machine Learning Service Provider
MPC	Multi Party Computation
OT	Oblivious Transfer
PPML	Privacy Preserving Machine Learning
PHE	Partial Homomorphic Encryption
SIMD	Single Instruction Multiple Data
SS	Secret Sharing
SWHE	SomeWhat Homomorphic Encryption

List of Figures

List of Tables

Part I

Introduction

I.1 Context and Problem Statement

Nowadays, a growing number of companies are migrating to business models that are based on collecting and leveraging users' data *(Data-Driven Business Model DDBM)*. This is possible by relying on Machine Learning (ML) methods to discover strategic and relevant information from this data. By following such a strategy, these companies fully capitalize their production process on the availability of data. However, these critical resources may simply not be available, or their quantity might not be sufficient to extract useful information.

Collaborative machine learning can be very useful in this context, where multiple parties with common interests can collaborate to obtain ML models with better accuracy using their joint data. Nevertheless, this scheme is not always feasible, due to legal, financial, and competitive constraints, especially when the manipulated data is considered to be sensitive. Such confidentiality problems can be observed in health systems, fraud detection, recommendation systems, etc.

Given the many benefits of collaborative learning, much research has been conducted to propose Privacy Preserving Machine Learning methods (PPML). These works fall into two main categories based on which privacy preservation technique is used, where some works rely on non-cryptographic techniques such as data randomization [1], and others employ cryptographic techniques such as homomorphic encryption [2] to ensure privacy preservation.

This thesis consists of two parts, in this first one, we consider a centralized ML as-a-service architecture where different data owners outsource their training data to a service provider that is supposed to do carry a training process on it without disclosing their content. This gave rise to our privacy preserving classification framework *PrivML* that we present below.

In a second part, we are interested in another more recent collaborative learning scheme which is Federated Learning (FL) where the different collaborators perform local training on their private data and only share model updates with an orchestrator who is responsible for aggregating the different model updates sent by the different collaborators, updating a global model and communicating it to them. This kind of scheme certainly allows not to share the private data of the participants with a third party. However, it remains vulnerable to a panoply of attacks [3] including poisoning attacks [4, 5] which we study in-depth, where we propose a new detection mechanism called *ARMOR* to mitigate them.

I.2 Improving the Efficiency of Cryptography-Based Privacy Preserving Machine Learning

Privacy Preserving Machine Learning solutions that rely on homomorphic encryption (HE) do not impact models' utility and accuracy while providing high privacy guarantees. That being said, homomorphic encryption is known to induce high computational overheads. Moreover, state-of-the-art HE-based PPML methods differ in terms of HE schemes (i.e., fully homomorphic FHE, somewhat homomorphic SWHE, or partially homomorphic encryption PHE), in terms of secure computation architecture (i.e., multi-party computation MPC vs. single-party computation SPC), and in terms of other techniques such as ciphertext packing, etc. [6]. This results in different impacts on computational overheads.

In this part of the thesis, we precisely explore the impact of different architectural and design choices of HE-based PPML methods on actual PPML performance.

Our main objective is to design and implement a privacy preserving framework that ensures a proper trade-off between the following criteria that we have identified to be crucial for outsourced PPML services' performance (i) *Privacy guarantees.* the provided solution must ensure end-to-end privacy preservation of training data, machine learning models, users' requests, and system responses; (ii) *Computational efficiency.* the computational overhead caused by privacy preservation mechanisms must be minimal; (iii) *Service utility.* the utility of the provided privacy preserving ML service must be as close as possible to their original implementations; (iv) *Service usability.* privacy preservation strategies must not affect the service usability and must provide a suitable user experience.

To achieve the goals described above, we propose *PrivML* an outsourced homomorphic encryption-based privacy preserving collaborative machine learning framework, that allows optimizing runtime and bandwidth consumption for widely used ML algorithms, using many techniques such as fast algorithms for large integer arithmetic, ciphertext packing, approximate computations, and parallel computing.

We evaluate *PrivML* using real-world datasets and compare it with the most relevant state-of-the-art HE-based PPML methods.

I.3 Mitigating Poisoning Attacks in Federated Learning

Federated learning (FL) is a new framework that enables orchestrated collaborative learning without explicit exchange of training data, thus improving the privacy of user data [7]. Due to its attractive guarantees, it has been rapidly adopted in many application domains such as next-word-prediction in Gboard [8], speech recognition [9], autonomous cars [10], and many others.

Nevertheless, despite its advantages, it has been shown that Federated Learning is very vulnerable to many attacks coming from the client-side since this framework is user-driven [11]. In this part of this thesis, we focused primarily on data and model poisoning attacks that target the robustness of Federated Learning [4, 5, 12]. In these attacks, adversaries attempt to inject a malicious task into the federated model along with its main task. This malicious task assigns a label chosen by the attacker to the input data with a specific trigger. For example, an attacker can bypass a facial recognition-based authentication system by assigning the wrong identity label to its images that authorize it to access the system.

Detecting poisoning attacks in federated learning is a challenging problem because participants only send model updates to the FL server instead of sending their raw training data. As a result, the FL server holds less information about users' behavior to detect malicious participants. Many mechanisms have been proposed in the state-of-the-art to enable the detection of such attacks. Although these mechanisms have various rules for detecting poisoning, they all rely on auditing the geometric shape of model updates sent by participants to the FL server.

In this second part of the thesis, we were able to show that attackers are still able to evade these detectors by fabricating model updates that mimic the updates of benign participants. Subsequently, we proposed *ARMOR*, a new GAN-based attack detector, to analyze the information that model updates capture about user data, instead of monitoring their geometric shapes.

We evaluate *ARMOR* using widely used image recognition datasets and deep neural network architectures and demonstrate that they out-perform state-of-the-art mechanisms in mitigating extremely aggressive poisoning attack scenarios.

I.4 Summary of Contributions

The contributions of this thesis are two folds : **(c1) : *PrivML*: An efficient outsourced online ML framework over encrypted data (c2) : *ARMOR*: A mitigation mechanism against poisoning attacks in federated learning.** In the following, we enumerate the publications, communications, and developed software prototypes in the scope of these two contributions.

I.4.1 Publications and Communications

- Fatma-Zohra El Hattab, **Rania Talbi**, Sara Bouchenak, and Vlad Nitu: *ARMOR*: Mitigating Poisoning Attacks in Federated Learning. *(Under Submission)*

- **Rania Talbi**, and Sara Bouchenak: How Practical is Cryptography-Based Privacy Preserving Outsourced Machine Learning *(Under Submission)*

- Fatma-Zohra El Hattab, **Rania Talbi**, Sara Bouchenak, and Vlad Nitu: Towards Mitigating Poisoning Attacks in Federated Learning. In: Conférence francophone d'informatique en Parallélisme, Architecture et Système (ComPAS). Lyon, France 2021.

- Jiyue Huang, **Rania Talbi**, Zilong Zhao, Sara Bouchenak, Lydia Y. Chen, Stefanie Roos: An Exploratory Analysis on Users' Contributions in Federated Learning. In: International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA). Virtual Conference 2020.

- **Rania Talbi**: Towards Practical Privacy-Preserving Collaborative Machine Learning at a Scale. In: IEEE-IFIP International Conference on Dependable Systems and Networks (DSN). València, Spain 2020.

- Fatma-Zohra El Hattab, **Rania Talbi**, Sara Bouchenak, and Vlad Nitu: How Effective Are Data Poisoning Attacks on Federated Learning. In: Conférence francophone d'informatique en Parallélisme, Architecture et Système (ComPAS). Virtual Conference 2020.

- Nassim Ait-Ali-Braham, **Rania Talbi**, Sara Bouchenak, Caroline Fontaine: Priv-ML: Leveraging Practical End-to-End Privacy in Collaborative ML Services. In: Conférence francophone d'informatique en Parallélisme, Architecture et Système (ComPAS). Anglet, France 2019.

- Rémi Canillas, **Rania Talbi**, Sara Bouchenak, Omar Hasan, Lionel Brunie, Laurent Sarrat: Exploratory Study of Privacy Preserving Fraud Detection. In: International Middleware Conference. Rennes, France 2018.

- **Rania Talbi**: Towards Incremental End-to-End Privacy Preserving Data Classification.In: International Middleware Conference. Doctoral Symposium. Rennes, France 2018.

- **Rania Talbi**, Sara Bouchenak, Lydia Y. Chen: Towards Dynamic End-to-End Privacy Preserving Data Classification. In: International Conference on Dependable Systems and Networks Workshops. Luxembourg City, Luxembourg 2018

The following communications took place during this thesis project:

- In ComPAS 2021 (06 - 09 July 2021), *ARMOR*: Towards Mitigating Poisoning Attacks in Federated Learning, virtual presentation.

- In DSN 2020 (29 June - 02 July 2021), Privacy-Preserving Collaborative Machine Learning at a Scale, virtual presentation.

- In IRYXIS 2019 - Winter edition, (02 - 05 December 2019), Towards Robust Federated Learning, Passau, Germany.

- In APVP 2019 (09 - 11 July 2019), Towards Practical Privacy-Preserving Collaborative Machine Learning at a Scale, Baie de Somme, France.

- In IRYXIS 2019 - Summer edition (18 - 20 June 2019), Towards Practical and Scalable Privacy-Preserving Collaborative Machine Learning Lyon, France.

- In GDR RSD ASF Winter School 2019 (04 - 07 February 2019), Towards Practical Privacy-Preserving Machine Learning, Grenoble, France.

- In Middleware 2018 (10 - 14 December 2018), Towards Incremental End-to-End Privacy Preserving Data Classification, Rennes, France.

I.4.2 Developed software

The following software prototypes were developed during this thesis:

- ***PrivML***: C++ library for outsourced privacy preserving machine learning over encrypted data.
  Available at: https://gitlab.liris.cnrs.fr/rtalbi/DAPPLE-2.0

- ***ARMOR***: Python library for poisoning attacks detection in federated learning using GAN-based class representatives generation.
  Available at: https://gitlab.liris.cnrs.fr/rtalbi/armorfd [1]

I.5 Thesis Roadmap

This manuscript is structured as follows. In Part II, we present our contribution in improving the efficiency of cryptography-based privacy preserving ML-as-a Service. Where we first provide a general state-of-the-art and background overview in Privacy Preserving Machine Learning (PPML). Then we present the design principles of our framework *PrivML* that addresses the high overhead issues in cryptography-based PPML methods. This part also provides the empirical evaluation results of *PrivML*. Part III is about robustness issues in federated learning, where we first start by providing some generalities on federated learning and the attacks that target it. After that, we discuss existing works in poisoning attacks detection in federated learning. We then present *ARMOR* a poisoning detection mechanism that we have proposed. Next, we present a detailed evaluation of *ARMOR* and assess its advantages and drawbacks. Part IV concludes this thesis and discusses future work and research directions.

[1] The *ARMOR* repository is not publically accessible for now considering that it is a work under submission.

Part II

PrivML: Improving the Efficiency of Cryptography-Based Privacy Preserving Machine Learning

II.1 Background on Privacy Preserving Machine Learning

Ubiquitous computing is continually generating vast amounts of data in the current information age, especially through IoT devices, web services, and social networks. The analysis of this data has shown to be highly beneficial to a countless number of services such as health-care [13], banking [14], cyber-security [15], commerce [16], transportation [17], and many others. However, much of the collected information may contain sensitive private data, which raises important privacy concerns [18, 19].

The main concern for users is that their personal and highly sensitive data such as photos and voice recordings are kept indefinitely by the companies that collect them and that they can neither delete them nor restrict the purposes for which they are used.

Moreover, even when individuals or entities trust each other and are willing to share their data, they might not be allowed to do so for legal reasons.

For example, let us consider the scenario in which many hospitals wish to jointly mine their patients' data for medical research. Usually, it is legally forbidden for these hospitals to pool their data or to reveal them to each other. As a consequence, classical machine learning solutions cannot be used. In such contexts, privacy constraints prevent valuable knowledge extraction. Also, the concerns over massive collection of data are naturally extending to analytic tools applied to data.

Machine Learning (ML), with its promise to efficiently discover valuable, non-obvious information from large datasets, is particularly vulnerable to malicious users through attacks against Machine Learning models [19, 20]. This concern is further amplified with the increasing popularity of cloud service providers and the paradigm of Machine Learning as a Service (MLaaS) [21].

To address the privacy issues in ML, a sub-field of Machine Learning, referred to as Privacy Preserving Machine Learning (PPML), has gained significant development in recent years. The objective of PPML is to protect sensitive information from unsolicited or unsanctioned disclosure, and meanwhile, preserve the utility of data and provide effective and efficient ML services. In short, Privacy Preserving Machine Learning aims to conciliate data exploration with privacy preservation.

Machine Learning might seem in total contradiction with privacy protection at first glance since the global purpose of the former is to extract valuable knowledge from raw data while ensuring privacy goes through limiting access to the data.

This inherent goal conflict in Privacy Preserving Machine Learning makes the problem extremely challenging. It has led to a rich literature with many heterogeneous solutions, relying on different techniques (secure multi-party computation, homomorphic encryption, differential privacy). However, despite the significant progress in this research area, the question of Privacy Preserving Machine Learning is far from being solved and there is still a large room for improvement of the current techniques.

This chapter provides a general overview of this emerging field. We start with a brief review of basic notions regarding machine learning and privacy preservation, followed by a taxonomy of PPML solutions and the field's core concepts, and conclude with a summary defining the research gap addressed in this first part of the manuscript.

II.1.1 Background on Machine Learning

Machine learning (ML) studies algorithms and statistical models that computer systems use to efficiently perform a specific task without explicit instructions, relying on patterns and inference instead. In other words, Machine learning algorithms build a mathematical model of sample data to make predictions or decisions without being explicitly programmed to perform them [22].

Machine learning algorithms differ in their global approach, the type of data they input and output, and the type of problems they were designed to solve.

The most straightforward and widely accepted classification of ML algorithms encompasses three main categories, which are: *Supervised Learning*, *Unsupervised Learning*, and *Reinforcement Learning* [23].

- **Supervised Learning** they are methods in which a training set is used and has both the input data and the respective desired label/output. This dataset allows the ML algorithm to learn specific patterns, distinguish data, and perform predictions on previously unseen and unlabelled records.

- **Unsupervised Learning** these techniques attempt to find relations in the data from unlabelled sets. They are used to find structure in data, like clustering data points or dimensionality reduction [24].

- **Reinforcement Learning** these algorithms allow an agent required to take automated decisions to learn adequate policies to achieve its goal and maximize the rewards it gets in a controlled environment when making good decisions.

According to [25], ML tasks can be decomposed into the following classes:

- **Association rule learning** a rule-based ML method is used for discovering interesting relations between variables in large databases. For example, a supermarket can use association rule mining to determine which products are frequently bought together and use this information for marketing purposes [26].

- **Clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters).
- **Classification** is the problem of identifying to which of a set of categories a new observation belongs. In order to perform this mapping, a function or a model is learned by deriving the relationship between the objects in the training set and their corresponding classes.
- **Regression** is the task of approximating a mapping function f from input variables to a continuous output variable y.

In this manuscript, we are particularly interested in supervised ML tasks, which are classification and regression.

II.1.2 Background on Privacy Preservation

In the information scope, Westin et al. [27] defined privacy as: "The claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others."

In other words, it is the right to control the handling of one's information. Thus, one can conclude that the main idea of information privacy is controlling the collection and handling of one's data.

Data privacy is often seen as an aspect of data security. This is a questionable claim since the goals of the two fields are quite divergent.

On the one hand, security protects the data against unauthorized access and modification. However, once the data reaches an authorized recipient, security does not impose any additional constraint having to do with revealing an individual's personal information. This is, on the other hand, the goal of data privacy.

Therefore, it is more appropriate to describe the relationship between data security and data privacy, as the former being a prerequisite of the latter. Data must be protected in storage and transmission by data security methods, but if one wants to achieve data privacy, additional steps must be taken to protect sensitive data.

II.1.2.1 Privacy Preserving Machine Learning (PPML)

Privacy Preserving Machine Learning (PPML) [28,29] is a relatively novel research direction where machine learning algorithms are analyzed for the side-effects they incur on data privacy. The main objective of privacy preserving machine learning is to develop methods for modifying the original data and algorithms in some way so that the private data and personal information remain secret even when applying ML algorithms [30].

II.1.2.2 A Taxonomy of Privacy Preserving Machine Learning Methods

Many taxonomies of PPML techniques have been proposed in the literature [30–32]. Based on these taxonomies, in the rest of this section, we present the most important dimensions to take into consideration when designing a PPML method, as summarized in Figure II.1.1.

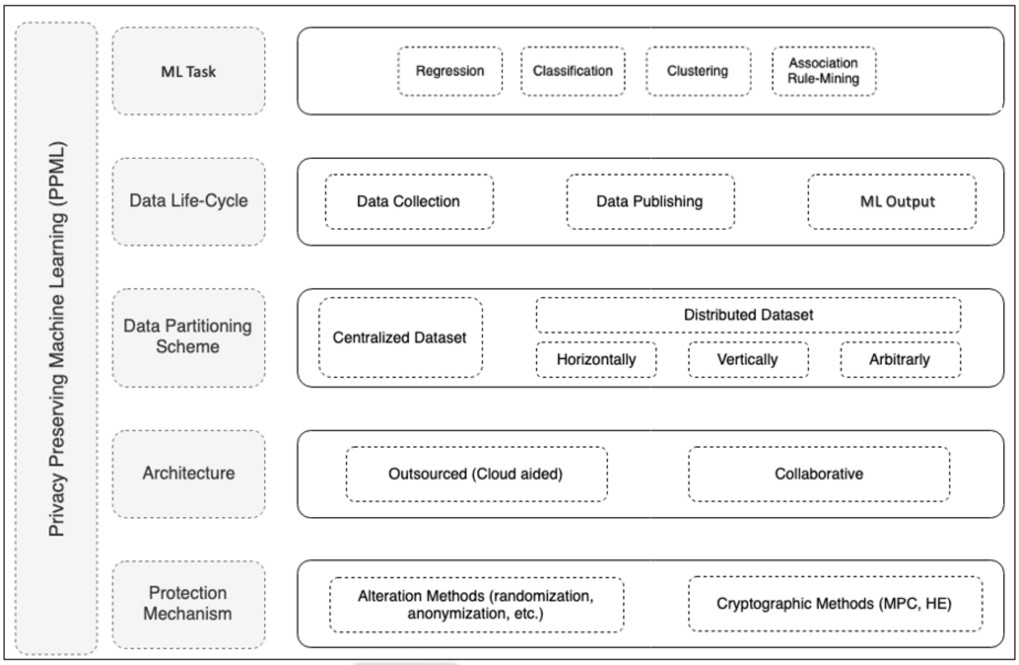


Figure II.1.1: A taxonomy of Privacy Preserving Machine Learning.

ML Task

All the ML tasks presented in the previous section might require a privacy protection mechanism in the presence of sensitive data. Some PPML techniques can be applied generically, regardless of the underlying ML task, while others are specifically designed for a specific ML task (e.g., association rule hiding).

Data Life-Cycle

Data goes through a series of steps during the data analysis process, which we often refer to as the data life-cycle. Each of these steps may include different actors that we can choose to trust or not. Therefore, it becomes necessary to design PPML techniques that ensure privacy during each phase. We briefly describe each of these phases below.

- **Data Collection** this is the step during which data is collected from data owners. Deploying privacy mechanisms in this phase assumes that the entity collecting the data is not trusted. This is generally achieved by adding randomization or an encryption mechanism to the sensory device that collects the data.

- **Data Publishing** this is the step during which the collected data will be released publicly or to third parties for data analysis. Therefore, the challenge is to give access to a dataset without disclosing sensitive information.

- **ML Output** the output of the ML algorithm can be extremely revealing, even without explicit access to the original dataset. In fact, it has been shown that Machine Learning algorithms are vulnerable to numerous types of attacks, for example, the membership inference attack proposed by Shokri et al. in [20].

Data Partitioning

Depending on the use case, data can be either centralized or partitioned across multiple data owners. The partitioning can be horizontal, vertical, or arbitrary.

System's Architecture

There are two major and widely used architectures in Privacy Preserving Machine Learning applications, which are the collaborative PPMLs and the outsourced PPMLs (See FigureII.1.2). We briefly explain both of the architectures below.

- **Collaborative PPMLs** this is the case where there are n entities who want to perform some ML task on the union of their datasets without revealing their private data. Thus, these parties have to collaborate, following a specified protocol, to achieve the desired task in a privacy preserving manner.

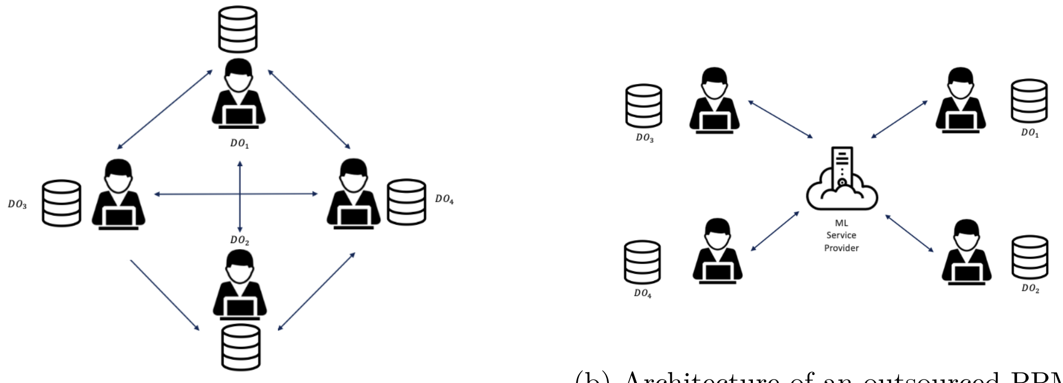
- **Outsourced PPMLs** this can be seen as an example of Machine Learning as a Service (MLaaS), where a service provider offers ML services to users. In these architectures, users and data owners have no reason to trust the service provider. Therefore, data owners might be reluctant to share sensitive information with the service provider, and the same holds for users who might require some protection on their queries and outputs.

Privacy Preservation Mechanisms

The mechanisms used to ensure privacy can vary significantly for the same ML task (e.g., perturbation techniques, secure multi-party computation, homomorphic encryption, etc.). These techniques differ in the privacy guarantees and performance they offer.

II.1.2.3 Basic Concepts of PPML

In this section, we present the different actors in a PPML system, the different types of attributes, and the possible information disclosures.



(a) Architecture of a collaborative PPML.

(b) Architecture of an outsourced PPML.

Figure II.1.2: Comparison between outsourced and collaborative PPMLs.

Actors

In a typical data analysis scenario, one can identify many actors that may be involved in different stages of the data analysis process. We list the most important ones below.

- **Data Owner (DO)** these are individuals or organizations that generate the original raw data and offer the data to others, either actively or passively.
- **Data Collector** it is a user or an organization who collects data from data owners and then publishes the data to an ML service provider.
- **ML Service Provider** is the entity who is responsible for performing ML tasks on the collected data.

Attributes Types

In general, we can divide the attributes of a given data record into three categories according to their semantics, which we list below.

- **Explicit Identifiers** these are the unique attributes that unambiguously identify an individual, such as the full name and the passport number.
- **Quasi-Identifiers** these attributes do not directly identify a person but provide partial information that may allow to re-identify it when gathered together with the assistance of other information, such as age, career, postcode, and so on.
- **Sensitive Information** these attributes contain private information about an individual, which should not be revealed (for example, medical data of a patient).

II.1.2.4 Evaluation Metrics of PPML Methods

Privacy Preserving Machine Learning algorithms can be evaluated according to different metrics, including the privacy level, the utility of the data, and the method's complexity.

Privacy the most obvious metric on which a PPML method must be evaluated is the level of privacy that it offers. We divide the privacy aspect into two components, namely *Data Privacy* which is measured by the ability of an attacker to reconstruct the original data. The more difficult it is, the better is the protection, and *Result Privacy* which consists of protecting ML models against attacks that use them to infer sensitive information about data.

Utility this metric evaluates the impact that PPML methods have on model accuracy. It is very common for PPML methods to alter training data by inserting false samples or blocking data values to hide sensitive information. They can also apply distortion on the output of the ML algorithm, which can negatively impact ML models' quality.

Complexity this metric relates to the *efficiency* and *scalability* of the considered PPML mechanism time and space-wise.

II.1.3 Related Work on Non-Cryptographic Privacy Preserving Machine Learning Techniques

Non-Cryptographic PPMLs are the most widely used lightweight methods to preserve privacy in the context of machine learning. These PPML techniques rely on a set of sanitization primitives that we enumerate below.

- **Generalization** is the replacement of a value by a more general one. For instance, numerical data may be defined by intervals.

- **Suppression** is the removal of a specific attribute to prevent disclosure (typically the case of explicit identifiers) or a specific row in a dataset.

- **Anonymization** is the de-association of Quasi-Identifiers (QIDs) and sensitive attributes in two separate tables making it more difficult to link them.

- **Randomization** is the replacement of original data with synthetic values that have identical statistical properties. For example, noise addition and data swapping.

Non-cryptographic PPML methods can be classified into three approaches which we discuss in the following.

II.1.3.1 Perturbation Approaches

The general paradigm of perturbation approaches can be described as follows. Given a set of data records denoted by $T = \{x_1, ..., x_N\}$. For each record $x_i \in T$, we apply a determined modification or perturbation to keep its true value secret.

Most perturbative methods reviewed below are special cases of matrix masking. If the original dataset is T, then the masked dataset T' is computed as, $T' = ATB + C$ where A

is a record-transforming mask, B is an attribute-transforming mask, and C is a displacing mask (noise).

Additive Noise

The additive noise method was first introduced in the context of Privacy Preserving Machine Learning in [28]. This technique randomizes the dataset by adding noise to it. To be more precise, for each record $x_i \in T$, a random noise component y_i is independently drawn from a noise distribution F_Y and then added to x_i. Thus, the new set of distorted records z_i is denoted by $T' = \{z_1 = x_1 + y_1, ..., z_N = x_N + y_N\}$.

Let X be the random variable denoting the data distribution of the original records, Y be the random variable describing the noise distribution, and Z be the random variable denoting the final records, we have, $Z = X + Y \implies X = Z - Y$ The distribution of Y is publicly known, whereas the distribution of Z has to be estimated using a kernel density estimation technique [33]. Then, by subtracting the density of Y, it is possible to reconstruct an approximation of X's density, as long as the number of samples N is large enough and the noise variance is not too high.

Randomization using additive noise is a simple technique that can be applied at the data collection phase and requires no intermediate storage of the real records. The downside is that outliers are more susceptible to adversarial attacks, requiring the addition of more substantial noise to hide them. It is also worth noticing that we only get the distribution of X, rather than actual records. Noise addition is vulnerable to many types of attacks, from which the most important are the *Dimensionality Reduction Attack*, and the *Maximum Likelihood Attack* [34].

Multiplicative Noise

Multiplicative noise follows some of the ideas of additive noise, with a crucial difference: the randomly generated noise is now multiplied by the attribute values instead of being added to them.

There have been many successful attempts to realize multiplicative perturbations, and most of them derive their roots in the work of [35].

Even though this method offers stronger privacy guarantees than additive noise, it remains vulnerable to a few types of attacks, from which the most important are *Known Input-Output Attack* and *Known Sample Attack* [34].

Data Swapping

Another widely used method to protect privacy is data swapping [36], in which the values across multiples records are swapped in order to enhance privacy. To preserve utility, it would be convenient to swap values that are close to each other. Therefore, swapping is controlled by the distance between the swapped values, meaning that close values are more likely to be swapped.

II.1.3.2 Group-Based Anonymization

An important drawback of the perturbative approaches is that they do not consider the case where a publicly available record can be used to infer the identity of its owner. Other frameworks such as K-Anonymity [37], L-Diversity [38], and T-Closeness [39] were proposed to ensure identity privacy preservation.

II.1.3.3 Machine Learning Output Privacy

Most of the Privacy Preserving Machine Learning (PPML) techniques we have presented so far are concerned with protecting individuals' privacy against an attacker who has direct access to the data. This is typically the case in a data publishing scenario. However, an attacker can also infer sensitive information without even accessing the training dataset. In fact, the outputs of ML algorithms can be highly revealing, even for an application that prevents access to raw data. An adversary may query such applications and infer sensitive information about the underlying data.

In the following, we provide a brief overview of the most important PPML algorithms designed to decrease the privacy leakage of ML models. It is also worth mentioning that all the techniques discussed in the previous sections can provide additional protection by first sanitizing the dataset before the data analysis process.

(ϵ)-Differential Privacy

The differential privacy framework was proposed by Dwork et al. in [40]. This framework offers strong privacy protection in the information-theoretic sense. The intuition is that an attacker may obtain expected information by multiple queries on a dataset based on his background knowledge regarding victims. Differential privacy ensures that the removal or addition of a single data record does not substantially affect the outcome of any analysis.
Definition a randomized function f gives differential privacy if for all datasets D_1 and D_2 differing on at most one element, and all $S \subseteq Im(f)$ [1], we have,
$P[f(D_1) \in S] < e^{\epsilon} P[f(D_2) \in S]$
A mechanism f satisfying this definition addresses concerns that any participant might have about the leakage of her personal information: even if the participant removed her data from the dataset, no outputs would become significantly more or less likely to occure. Differential privacy is achieved through noise addition to the output of the function f, usually using the Laplace mechanism [40].

[1] For a given function f, we denote $Im(f)$ as the range of f, that is: $Im(f) = \{f(x)|x \in D_f\}$, where D_f is the domain of f.

Simply stated, the output is distorted as follows: $T(x) = f(x) + Y$, where $Y \sim Lap(\lambda)$ is a Laplacian noise [2]. By doing so, we obtain a mechanism that is ϵ-differentially private with $\epsilon = \frac{\Delta f}{\lambda}$, where $\Delta f = \max_{D_1, D_2 \in D} \|f(D_1) - f(D_2)\|_1$ is often referred to as the sensitivity of the function f.

Association Rule Hiding

Association rule hiding is a privacy preserving technique introduced in [41] whose objective is to perform association rule mining without revealing the sensitive rules, by performing:

- **Distortion** in distortion [42], a given transaction is changed to a different value.
- **Blocking** where some entries of the dataset are blocked to prevent the discovery of specific sensitive association rules [43].

In both approaches, some of the non-sensitive rules can be lost, and new false rules (often referred to as *ghost* rules) may be created because of the distortion or blocking process. These side-effects are undesirable since they reduce the utility of the data.

Downgrading Classifier Effectiveness

Machine learning models, and particularly classifiers, are prone to many types of attacks, for instance, the membership inference attack [3] [20]. Furthermore, the results of a classification application may be a piece of sensitive information for the owner of a dataset. Therefore, we want to be able to modify the data so that the accuracy of the classification process is reduced while keeping it acceptable for most applications. An example of these works would be In [44] where the case of decision trees is considered. Note that the approaches used for association rule hiding can also be generalized to rule-based classifiers since rule-based classifiers often use association rule mining as a subroutine.

Query Auditing and Inference Control

There are two main approaches for query management in the literature of Privacy Preserving Machine Learning, which are *Query Auditing* and *Query Inference Control*. **Query Auditing** the basic idea is to perform some *filtering* on the model queries. In other words, a fraction of queries is denied to prevent the querier from gaining too much information about the raw data, consequently compromising the privacy of the training set records. Several algorithms were proposed to tackle this problem [45]. **Query Inference Control** in this approach, instead of controlling the set of authorized queries, perturbations are

[2] The probability density function of the Laplace distribution is given by the formula: $f(x|\mu, \lambda) = \frac{1}{2\lambda}\,\mathrm{e}^{(}-\frac{|x-\mu|}{\lambda})$, where μ is the mean and λ is a scale parameter.

[3] A membership inference attack is an attack against Machine Learning models that aims to determine if a particular record was part of the training set of the model.

used to protect the underlying data. These perturbations can be applied directly to the data [38, 39] or added to the query result.

II.1.4 Background on Cryptography

In this section, we introduce the most important cryptographic primitives that are widely used in Privacy Preserving Machine Learning (PPML). Generally speaking, even the most efficient of these techniques are considerably more expensive than the Non-cryptographic methods. Despite this, the cryptographic approach remains attractive since it provides much stronger privacy guarantees and does not alter data utility. There are two important families of techniques that were exploited by researchers over the years to implement cryptographic PPML methods, which are *Homomorphic Encryption* and *Secure Multi-Party Computation.*

II.1.4.1 Homomorphic Encryption

Homomorphic Encryption (HE) is a special kind of encryption that allows a third party (e.g., an ML service provider) to perform certain computable functions over encrypted data while preserving the features of these functions and the format of the encrypted data. This paradigm of computation over encrypted data was suggested by Rivest et al. in [46]. They imagined a scenario where a client encrypts its input x and sends it to a server, which can then evaluate a function f on the encrypted input. The server returns an encryption of the evaluated ciphertext to the client, who decrypts the output and recovers the result.

The encryption method must have certain special properties so that it allows the processing of encrypted data. For example, Rivest et al. observed in [46] that RSA [4] enables multiplication of encrypted values. Specifically, starting from the encryption of m_1 and m_2, one can compute an encryption of $m_1 m_2$ without knowing the secret key. This property is formally described in Equation (II.1.1), where N and e are the public parameters of the cryptosystem.

$$\begin{cases} c_1 = Enc_{pk}(m_1) = m_1^e \mod N \\ c_2 = Enc_{pk}(m_2) = m_2^e \mod N \end{cases} \implies c_1 c_2 = (m_1 m_2)^e \mod N = Enc_{pk}(m_1 m_2) \tag{II.1.1}$$

Rivest et al. naturally asked whether it was possible to compute more general functions over encrypted data, and what one can do with an encryption scheme that enables such computations. This question was the first stepping stone to what became one of the most active research topics in modern cryptography.

[4] RSA (Rivest–Shamir–Adleman) is one of the first public-key cryptosystems and is widely used for secure data transmission.

$$\begin{tikzcd}
c_1, c_2, \dots, c_t \arrow[r, "Eval"] & Eval(f, c_1, c_2, \dots, c_t) \arrow[d, "Dec(sk,.)"] \\
m_1, m_2, \dots, m_t \arrow[u, "Enc(pk,.)"] \arrow[r, "f"] & f(m_1, m_2, \dots, m_t)
\end{tikzcd}$$

Figure II.1.3: Main primitives of a HE scheme.

Definition a Homomorphic Encryption Scheme consists of four procedures (KeyGen, Encrypt, Decrypt, Eval), each of which is described below.

- KeyGen generates the cryptographic keys $(sk, pk) \leftarrow KeyGen(\lambda)$, where sk is the secret key, pk the public key and λ the security parameter [5].
- Encrypt produces a cipher $c \leftarrow Enc(pk, m)$ from a plaintext message m.
- Decrypt recovers the underlying plaintext message $m \leftarrow Dec(sk, c)$ from a ciphertext c.
- Eval allows computations over ciphers. Specifically, for $f \in \mathcal{F}$, where $\mathcal{F}$ is some class of functions, Eval outputs $c' \leftarrow Eval(pk, f, \vec{c})$ such that $Dec(sk, c') = f(\vec{m})$, where $\vec{c}$ is an encryption of $\vec{m}$.

An illustration of the evaluation procedure is given in Figure II.1.3. It shows that homomorphically computing the function over encrypted inputs gives the same result as encrypting the function's output applied to the plaintext inputs. Depending on the family of functions a homomorphic cryptosystem can evaluate, we distinguish three main types of homomorphic schemes, which are Partially Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SWHE), and Fully Homomorphic Encryption (FHE). In the following, we introduce the three HE families, their properties, and examples of state-of-the-art schemes for each category.

Partially Homomorphic Encryption (PHE)

A Partially Homomorphic Encryption (PHE) scheme allows *some* computation over encrypted data. More precisely, a PHE either supports addition or multiplication, but not both. Depending on the supported operation, one can distinguish two sub-families of PHE, namely additive and multiplicative homomorphisms.

- `Additively Homomorphic Encryption` these schemes support the addition of ciphertexts only. Formally, for a certain operation $\oplus$ (usually multiplication), we have,

[5] In cryptography, a security parameter is a way of measuring of how "hard" it is for an adversary to break a cryptographic scheme.

$E(m_1) \oplus E(m_2) = E(m_1 + m_2)$. An example of such a scheme is the well-known Paillier cryptosystem [47]. It is also worth mentioning that any additive homomorphic cryptosystem allows performing multiplications with unencrypted scalar values, i.e, $E(m_1 r) = E(m_1) \otimes r = E(m_1) \oplus \underbrace{...}_{r\ times} ... \oplus E(m_1)$.

- `Multiplicatively Homomorphic Encryption` in a multiplicatively homomorphic encryption scheme, it is possible to evaluate the encryption of the product of two messages from their encryptions. Stated differently, we have, $E(m_1) \otimes E(m_2) = E(m_1 m_2)$. Popular examples of such schemes are RSA [46] and ElGamal [48].

Table II.1.1 provides a non-exhaustive list of the most widely used Partially Homomorphic Cryptosystems. All of them are public-key cryptosystems. They are quite limited in terms of homomorphic capabilities (i.e., the set of supported operations), but are more efficient than the two upcoming families.

Table II.1.1: A list of Partially Homomorphic Encryption Schemes.

Cryptosystem	Type	Message space	Hardness assumption
[46]	Mult	$\mathbb{Z}_N$	Factoring
[49]	Add	$\mathbb{Z}_2$	Quadratic Residuosity
[48]	Mul	$\mathbb{Z}_N$	Discrete log
[50]	Add	$\mathbb{Z}_N$	Higher Residuosity
[47]	Add	$\mathbb{Z}_N$	Composite Residuosity
[51]	Add	$\mathbb{Z}_N$	Decisional Diffie-Hellman and Factoring
[52]	Add	$\mathbb{Z}_N$	P-Subgroup Assumption
[53]	Add	$\mathbb{Z}_N$	Decisional Composite Residuosity

Somewhat Homomorphic Encryption (SWHE)

After the first plausible Fully Homomorphic Encryption scheme (FHE) published in 2009 [54], many Somewhat Homomorphic (SWHE) versions of FHE schemes were also proposed because of the performance issues associated with FHE. In fact, every FHE scheme is built upon a SWHE scheme to which a special procedure called *bootstrapping* is added.

This section focuses on major SWHE schemes developed before 2009, which were used as a stepping stone to the first plausible FHE scheme. We leave the *new generation* of SWHE schemes to the next section since they are inseparable from FHE cryptosystems.
An SWHE scheme supports both additions and multiplications but in a bounded number. Indeed, this kind of encryption scheme is limited to evaluating low-degree polynomials over encrypted data. In other words, the number of possible additions and multiplications over a ciphertext is limited.
In the literature of Homomorphic Encryption schemes, one of the first SWHE cryptosystems

is the Polly Cracker scheme [55]. It allows both multiplication and addition operations over ciphertexts. However, the ciphertext size grows exponentially with homomorphic operations, and the multiplication operation is very costly. Later more efficient variants [56, 57] are proposed, but almost all of them are later shown vulnerable to attacks [58, 59]. Therefore, they are either insecure or impractical.

One of the most important steps toward FHE was introduced by Boneh-Goh-Nissim (BGN) in [60]. BGN evaluates 2-DNF [6] formulas on ciphers, and it supports an arbitrary number of additions and one multiplication by keeping the ciphertext size constant.

Another idea of evaluating operations on encrypted data is realized over different sets. Sander, Young, and Yung (SYY) described the first SWHE scheme over a semigroup, NC^1 [61] [7]. The proposed scheme supported polynomially many *ANDing* of ciphertexts with one OR/NOT gate. However, the ciphertext size increased by a constant factor with each OR/NOT gate evaluation. Yuval Ishai and Anat Paskin (IP) [62] expanded the set to branching programs, which are the directed acyclic graphs where every node has two outgoing edges with labeled binary 0 and 1.

Table II.1.2 summarizes the most important Somewhat Homomorphic Encryption Schemes proposed before 2009.

Table II.1.2: A list of Somewhat Homomorphic Encryption Schemes.

Cryptosystem	Circuit Size	Circuit Type
SYY [61]	Poly-many AND and one OR/NOT	NC^1 circuit
BGN [60]	Unlimited Add and 1 Mul	2-DNF formula
IP [62]	Arbitrary	Branching Programs

Fully Homomorphic Encryption (FHE)

An encryption scheme is fully homomorphic if it can evaluate any boolean circuit without restriction on the depth or the number of gates. The existence of a Fully Homomorphic cryptosystem remained an open question for 30 years until Craig Gentry's breakthrough in 2009 [54]. In the following, we present the simplified paradigm behind FHE cryptosystems and the recent developments in the field.

The noise issue in his breakthrough, Gentry [54] started by proposing a semantically secure SWHE based on ideals [8]. The idea was that plaintext values are perturbations of an element of the ideal, such that it is impossible to compute the nearest ideal element to a cipher without the secret key. In order for the scheme to be secure, the perturbation needs to be a random noise added to the cipher. The decryption function removes this noise and

[6] DNF stands for Disjunctive Normal Form, that is a boolean expression with at most two literals in each clause.

[7] NC^1 is a complexity class that includes circuits with poly-logarithmic depth and polynomial size.

[8] An ideal is simply a subset of a ring with some interesting algebraic properties.

treats each point as if it were located in the nearest unperturbed location. However, when homomorphic operations are applied to the ciphertexts, their noise grows. The perturbed point thus wanders far away from its actual position until decryption starts returning a wrong value.

Bootstrapping the noise growth dramatically limits the set of functions that can be computed homomorphically. Hence, the construction described above is a Somewhat Homomorphic Encryption (SWHE) scheme. Gentry proposed a solution to this problem by extending the SWHE with a particular procedure, which he called *bootstrapping*. On a very high level, bootstrapping is a general procedure that takes as input a ciphertext and outputs a new encryption of the same underlying plaintext value but with much less noise.

Bootstrapping is a technique in which a cryptosystem evaluates its decryption function homomorphically.

Evolution of FHE Schemes although Gentry's scheme [54] was very promising, it also had many bottlenecks, such as its computational cost in terms of applicability in real life. Therefore, many new schemes and optimizations have followed his work to address the bottlenecks mentioned above.

After Gentry's work, lattices have become more popular among cryptography researchers. First, works like Smart and Vercauteren [63] focused on just improving Gentry's scheme in [54]. Then, an FHE scheme over integers based on the Approximate-GCD problem was introduced by Van Dijk et al. [64].

Afterward, another FHE scheme whose hardness is based on Ring Learning with Error (RLWE) problem was suggested in [65]. Lastly, an NTRU-like FHE was presented for its promising efficiency and standardization properties [66]. NTRU-Encrypt is an old and strongly standardized lattice-based encryption scheme whose homomorphic properties were realized recently. So, these and similar attempts can be categorized into under four main FHE families: (1) ideal lattice based [54], (2) over integers [24], (3) (R)LWE based [67], and (4) NTRU-like [66].

For simplicity, we present a version of FHE schemes over integers, which is already the most simple yet the least efficient compared to other modern FHE schemes.

Fully Homomorphic Encryption over Integers these schemes operate over integers and their hardness assumption is based on the Approximate-Greatest Common Divisor (AGCD) problem [68] [9]. The primary motivation behind the scheme is its conceptual simplicity. We briefly present a symmetric version of a FHE over integers.

- KeyGen for the given security parameter λ, a random odd integer k of bit length μ is generated.
- Encrypt for random large prime numbers p and q, choose a small number $r << k$. Then, the message $m \in \{0, 1\}$ is encrypted by $c = E(m) = m + 2r + pq$, where p is

[9] AGCD problem tries to recover a secret integer p from the given set of $x_i = pq_i + r_i$.

kept hidden as a private key and c is the ciphertext.

- DECRYPT the ciphertext can be decrypted as follows: $m = D(c) = (c \mod p) \mod 2$. Decryption works properly only if $m + 2r < \frac{p}{2}$. This limits the depth of the homomorphic operations performed on the ciphertext.

- ADD $E(m_1) + E(m_2) = m_1 + 2r_1 + pq_1 + m_2 + 2r_2 + pq_2 = (m_1 + m_2) + 2(r_1 + r_2) + (q_1 + q_2)p$. The output clearly falls within the ciphertext space and can be decrypted if the noise $|m_1 + 2r_1 + m_2 + 2r_2| < \frac{p}{2}$, where p is the private key. Since r_1, $r_2 << p$, a various number of additions can still be performed on ciphertext before noise exceeds $\frac{p}{2}$.

- MUL. $E(m_1)E(m_2) = (m_1 + 2r_1 + pq_1)(m_2 + 2r_2 + pq_2 = m_1m_2 + 2(m_1r_2 + m_2r_1 + 2r_1r_2) + kp$. The encrypted data can be decrypted if the noise is smaller than half of the private key, i.e., $|m_1m_2 + 2(m_1r_2 + m_2r_1 + 2r_1r_2)| < \frac{p}{2}$. The noise grows exponentially with the multiplication operation. This puts more restriction over the homomorphic multiplication operation than addition.

Leveled Homomorphic Encryption a Leveled scheme is a Somewhat Homomorphic Encryption scheme (SWHE) with good noise management techniques. This terminology was introduced in [69], and any recent practical SWHE scheme is, in fact, a leveled one. As previously mentioned, any FHE scheme has an associated SWHE cryptosystem (by simply removing bootstrapping). It turns out that it is possible to evaluate non-trivial circuits for a leveled scheme without resorting to bootstrapping. This idea has led to many PPML solutions with improved efficiency (especially for the prediction phase of Machine Learning models).

Packing one very important feature that many state-of-the-art homomorphic cryptosystems provide is *packing* or *SIMD* (Single Instruction Multiple Data). Packing is a feature that allows encoding many plaintext slots into a unique ciphertext. Subsequently, the operations performed over this cipher automatically translate to the individual slots, thereby allowing to process batches of data without any additional cost. For modern schemes, one can batch thousands of values into the same cipher, which provides important performance speed-up during the computations. As we will see in the next chapter, packing is one of the most critical features that allowed the development of practical PPML solutions using Homomorphic Encryption.

II.1.4.2 Secure Multi-Party Computation (MPC)

Secure Multi-Party Computation is a sub-field of cryptography allowing multiple parties to collaboratively compute a function over their inputs while keeping them private [70]. Unlike traditional cryptography, where the adversary is outside the system of participants, the adversary in this model controls actual participants. In the following, we briefly review some definitions and basic notions around the paradigm of secure computation.

Trusted Party

The easiest way for a set of parties to achieve the secure computation of arbitrary functions over their private inputs is to delegate the computation to a trusted party. Indeed, in the real world, there is no such trusted party. Therefore, participants cannot afford the risk of revealing their inputs to a third party. MPC protocols seek to offer the same security guarantees as a trusted party model without requiring such a strong assumption.

Security in MPC

Defining security in the context of Secure Multi-Party Computation is a complex task. Certainly, there are many desirable properties that one would want every secure MPC protocol to satisfy. According to [70], the most important of these requirements are:

- **Privacy** no participant should be able to learn anything about other parties' inputs, except what can be derived from the output.
- **Correctness** every party should receive the correct output of the computation.
- **Independence of Inputs** corrupted parties must choose their inputs independently of the honest parties. This property can be crucial in specific contexts, for example, in a sealed auction.
- **Fairness** if one party learns the output, then all the parties learn it as well.

Note that the above list is not meant to be exhaustive, and defining security as a set of requirements is not a viable approach. Therefore, to provide formal security guarantees, it is necessary to give a simple and general definition of security that can implicitly capture all the possible requirements. This is achieved using the standard Real World/Ideal World Paradigm [71].

Adversarial Model

The MPC framework requires a very rigorous approach. Among the parameters that need to be explicitly specified, the most important one is the adversarial model. An adversary is an entity that takes over a subset of the parties and wants to attack the protocol. The parties controlled by the adversary are called corrupted and follow the adversary's instructions.

Various aspects need to be defined when considering a given adversary. In the following, we describe the main types of adversaries encountered in the literature.

- **Corruption strategy** the corruption strategy specifies when the parties are corrupted. The most widely used models are:
  - `Static corruption` in this model, the adversary chooses a fixed subset of corrupted parties before the beginning of the computation.
  - `Adaptive corruption` in this model, the adversary is given the ability to corrupt parties dynamically during the computation.
- **Allowed adversarial behavior** specifies the set of actions corrupted parties are allowed to take. The two main models in the MPC literature are:
  - `Semi-Honest Adversary` in this model, the corrupted parties do not deviate from the specified protocol. However, the adversary can view the private data of all the corrupted parties and attempts to use this to learn information that should remain private.
  - `Malicious Adversary` in this adversarial model, the corrupted parties can arbitrarily deviate from the protocol specification, according to the adversary's instructions. This is a relatively strong adversarial model since it allows any malicious behavior.
- **Complexity** finally, one should also formalize the computational power of the adversary. Two types of adversaries are generally considered:
  - `Polynomial-time` the adversary is allowed to run in (probabilistic) polynomial-time. We remark that any attack that cannot be carried out in polynomial-time is not a threat in real life.
  - `Computationally unbounded` in this model, the adversary has no computational limits whatsoever.

II.1.4.3 Universal Primitives of MPC

We introduce the most widely used primitives in Secure Multi-Party Computation, namely *Oblivious Transfer*, *Secret Sharing*, *Garbled Circuits.* These building blocks are general and allow the computation of arbitrarily complex functions securely.

Oblivious Transfer

Oblivious Transfer (OT) [72] is arguably the most essential primitive in secure multi-party computation. In fact, it was shown by Kilian et al. [73] that using only oblivious transfer; it is possible to construct any secure multi-party protocol.

Definition a *1-out-of-n oblivious transfer* involves two parties, a sender and a receiver. At a very high level, the functionality of the protocol can be described as follows. The sender transfers one of potentially many pieces of information to a receiver but remains oblivious as to what piece (if any) has been transferred, as shown in Table II.1.3.

Table II.1.3: Oblivious Transfer inputs and outputs.

	Sender	Receiver
Input	$x_1, ..., x_n$	$i \in 1, ..., n$
Output	Nothing	x_i

1-out-of-2 Oblivious Transfer. we describe a simplified version of the general *1-out-of-n oblivious transfer*, namely the *1-out-of-2 oblivious transfer* introduced by Even et al. [74]. In this variant, the sender only holds two messages m_0 and m_1.

Algorithm 1 A simple 1-out-of-2 oblivious transfer protocol

```
Inputs: Messages $m_0$ and $m_1$                      Inputs: $\sigma \in \{0, 1\}$
Outputs: $\emptyset$                                   Outputs: $m_\sigma$

                                                         Generate a pair of (secret, public) keys
                                                         $(K_\sigma, P_\sigma)$ and a random $P_{1-\sigma}$
                                                         Send $(P_\sigma, P_{1-\sigma})$
  $(c_0, c_1) \leftarrow (Enc_{P_0}(m_0), Enc_{P_1}(m_1))$
  Send $(c_0, c_1)$
                                                         Decrypt $c_\sigma$
```

The simple protocol described in Algorithm 1 is a valid instantiation of 1-out-of-2 oblivious transfer. In fact, by assumption, the sender should not be able to distinguish between P_σ and $P_{1-\sigma}$. The receiver, on the other hand, does not know $K_{1-\sigma}$ since $P_{1-\sigma}$ was randomly generated. Therefore, he will only be able to decrypt c_σ.

Secret Sharing

Secret sharing is also a fundamental building block in secure multi-party computation, which was introduced independently by Shamir [75] and Blakley [76]. Secret sharing refers to a set of methods for distributing a secret amongst a group of n participants; each allocated a share of the secret. The secret can be reconstructed only when a sufficient number $t \leq n$ of shares are combined together. The party that distributes the secret is called *the dealer*, and the others are called *the players.*

Definition a t-out-of-n secret sharing scheme over message space $\mathcal{M}$ is a pair of algorithms (`Share, Reconstruct`) such that:

- `Share` is a randomized algorithm that on any input $m \in \mathcal{M}$ outputs a n-tuple of shares $(s_1, ..., s_n)$.

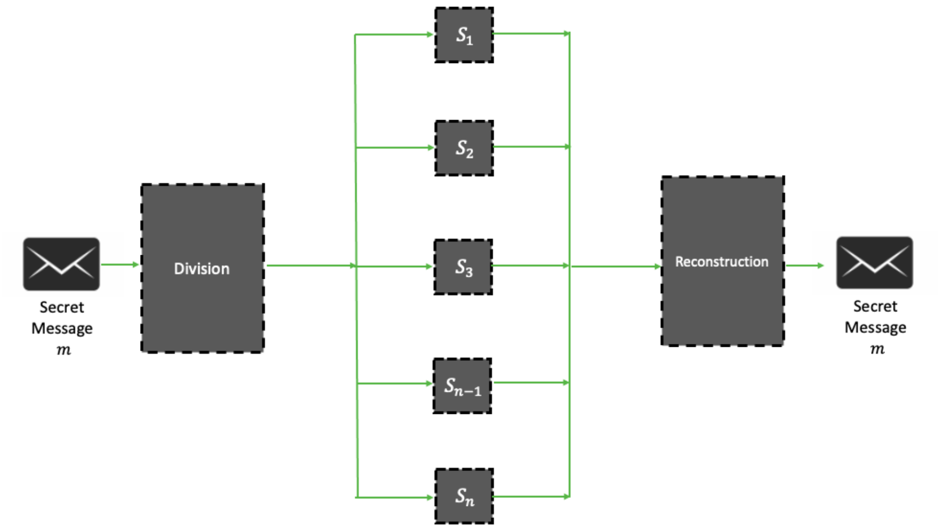


Figure II.1.4: Secret Sharing diagram.

- `Reconstruct` is a deterministic algorithm that given a t-tuple of shares outputs a message in $\mathcal{M}$

This scheme satisfies the following correctness requirement: $\forall m \in \mathcal{M}, \forall S = \{i_1, ..., i_t\} \subseteq \{1, ..., n\}$ *of size t*, $\underset{\texttt{Share}(m)\to(s_1,...,s_n)}{P}[\texttt{Reconstruct}(s_{i_1}, ..., s_{i_t}) = m] = 1$

Many different and complex secret sharing schemes were developed in the literature. In this section, we briefly introduce the one proposed by Shamir and al. in [75].

Shamir's Secret Sharing Scheme The idea behind Shamir's threshold scheme is that k points are sufficient to define a polynomial of degree $k-1$. Thus, the shares are simply evaluations of a randomly generated polynomial at different points. The protocol for generating the shares is described in Algorithm 2.

Algorithm 2 Shamir's Share Computation

Inputs: Secret m,
Outputs: Shares $s_1, ..., s_n$

```
$a_0 \leftarrow m$
Uniformly generate $a_1, ..., a_n$
Construct the polynomial $f(x) = a_0 + a_1 x + ... + a_{k-1} x^{k-1}$
Evalute $s_i = f(i)$ for $i \in [1, n]$
Send $s_i$ to $P_i$ for $i \in [1, n]$
```

The secret reconstruction follows immediately from Lagrange's formula [77]. Given k points $(x_0, y_0), ..., (x_{k-1}, y_{k-1})$, there is a unique polynomial f of degree $k-1$ such that $y_i = f(x_i)$ and is given by formula II.1.2.

$$f(x) = \sum_{i=0}^{k-1} y_i l_i(x) \tag{II.1.2}$$

where $l_i(x)$'s are the Langrange's basis polynomials defined in Formula II.1.3.

$$l_i(x) = \prod_{0 \leq m \leq n-1, m \neq i} \frac{x - x_m}{x_i - x_m} \tag{II.1.3}$$

Once the polynomial is reconstructed, the parties simply need to compute $m = f(0)$.

Secure Computation with Secret Sharing. secret sharing can be used to perform arithmetic operations over shares. Starting from the shares of A and B, it is possible to privately compute shares of $A + B$ and AB'. Therefore, one can evaluate arbitrary polynomials over shares in a privacy preserving manner. We briefly highlight how this applies to Shamir's Secret Sharing Scheme.

- **Addition** the shares of a sum are simply the sums of the shares from the additive properties of polynomials.
- **Multiplication** is not as straightforward as addition and requires communication between the involved parties. The complete algorithm can be found in [78].

Garbled circuits

Yao's garbled circuits [79] are a family of secure two-party computation protocols that allow two parties, say P_1 and P_2, each of which has its own private inputs (x and y respectively), to jointly compute a function $f(x, y)$ (expressed as a Boolean circuit) without revealing their secret inputs.

At a very high level, the protocol can be described as follows:

1. The underlying function is described as a Boolean circuit with 2-input gates. The circuit public.
2. P_1 garbles (encrypts) the circuit. We call P_1 the garbler.
3. P_1 sends the garbled circuit to P_2 along with his encrypted input.
4. P_2 through oblivious transfer receives his encrypted inputs from P_1.
5. P_2 evaluates (decrypts) the circuit and obtains the encrypted outputs. We call P_2 the evaluator.
6. P_1 and P_2 communicate to learn the output.

II.1.5 Related Work on Cryptography-Based Privacy Preserving Machine Learning

Cryptography-based Privacy Preserving Machine Learning approaches seek to protect sensitive information from unauthorized parties. For example, ideally, both storage and computation over data would be outsourced in an outsourced classification service. Allowing a service provider to access training data or even classification requests and responses can raise critical privacy concerns in these settings. Moreover, the model itself might be sensitive as well. Figure II.1.5 illustrates the sensitive parts of an outsourced Machine Learning application.

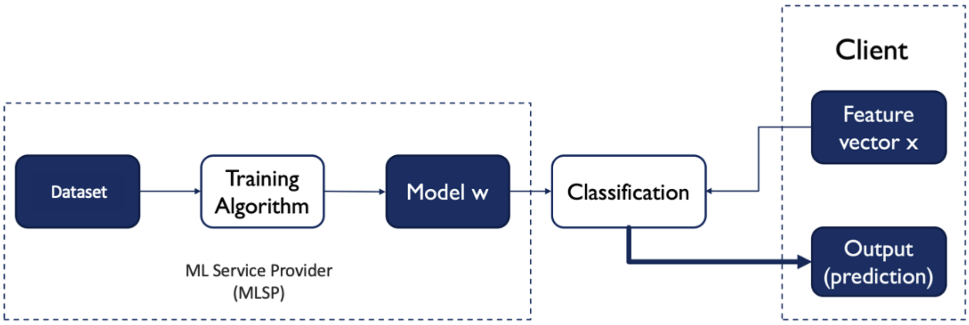


Figure II.1.5: Sensitive information in a Machine Learning workflow. Each shaded box indicates private data that should be accessible to only one party.

In a collaborative PPML solution where no external service provider is implied, the ML task is run by a set of parties that do not trust each other but still want to collaborate and gather common knowledge on their joint datasets.

These are the two most important scenarios that are covered by cryptography-based PPML methods. However, as explained in Section II.1.2.2, there are many other variables to be considered, for instance: **(1) which Machine Learning algorithm(s) to target?; (2) which phase of the algorithm? training, classification, or both?; (3) what cryptographic techniques to use?; (4) what is the required security level? which information leakage can be afforded?**

This important number of parameters and recent advances in cryptography have led to the development of a rich literature around the field of Privacy Preserving Machine Learning. In the following, we present a list of some of the most relevant state-of-the-art works that adopted cryptographic primitives to ensure privacy preservation.

We arrange the selected works based on the cryptographic primitives they rely on to ensure privacy, namely Homomorphic Encryption (HE), Secure Multi-Party Computation (MPC), or a mix of both.

II.1.5.1 Homomorphic Encryption-Based PPML Methods

This section encompasses PPML methods that are solely based on Homomorphic Encryption. These solutions are arguably the most expensive ones regarding the computational overhead but generally offer the highest level of security.

ML Confidential

Graepel et al. [80] suggested the use of Homomorphic Encryption for ML algorithms. They focused on designing protocols to train ML models over encrypted data using a *Somewhat Homomorphic Encryption* scheme (SWHE) and therefore were forced to use learning algorithms in which the training phase can be expressed as a low degree polynomial. As a result, most of the algorithms proposed were of the Linear Means (LM) classifier and Fisher's Linear Discriminant (FLD) Classifier. Their solution also supported private inference.

This work was the first attempt to train Machine Learning models using solely Somewhat Homomorphic Encryption (SWHE). They managed to build encrypted models from encrypted data on a small scale. Nevertheless, efficiency degrades rapidly as the size of the dataset grows (either horizontally or vertically). The authors reported that their solution was roughly six orders of magnitude slower than plaintext training and that the overall accuracy was not significantly affected by encryption.

Privacy Preserving Naive Bayes

Despite its simplisity, the Naive Bayes classifier is widely used due to its effectiveness. It is even commonly adopted as a baseline standard by which other classifiers are evaluated.

Liu et al. [81] realized a secure patient-centric clinical decision support system based on naïve Bayesian classification. They used an additive homomorphic [82] proxy aggregation scheme to convert the encryption under different public keys into encryption under a unique public key. This approach requires much less communication than fully distributed solutions. However, this particular solution reveals the model (i.e., the table of probabilities) in plaintext to a third party, which is a critical privacy breach that might not be acceptable in many contexts.

Similarly [83] relies on Gentry's fully homomorphic cryptosystem [54] to implement a privacy-preserving Naive Bayes classifier under the assumption of honest-but-curious participants. In this solution, a data owner outsources her training data encrypted under the BGV cryptosystem to an ML service provider (MLSP). The latter is composed of two parties P_1 and P_2 that collaborate in a two-party computation protocol to train a Naive Bayes model using the homomorphic properties of BGV. This solution also covers the inference phase. Furthermore, the MLSP's two parties P_1 and P_2 are assumed to be non-colluding. In order to reduce its computational costs, this PPML method relies on a row-wise ciphertext packing where each data record is encrypted in the same ciphertext. However, the computational overhead remains high since the authors rely on an old generation FHE scheme.

Privacy Preserving Logistic Regression

In statistics, the logistic model (or logit model) is a widely used statistical model for classification. Its simplicity and computational tractability made it an interesting candidate for Privacy Preserving Machine Learning, especially since this model is extremely useful in the medical field and genomics [84].

Kim et al. [2] designed a solution for privacy preserving logistic regression using *Fully Homomorphic Encryption* [10]. The authors used the cryptosystem developed by Cheon et al. [85], which supports approximate arithmetic of encrypted messages. Unlike existing methods, this cryptosystem trades precision for efficiency so that the size of parameters does not grow too large and is therefore very suitable for Machine Learning. The authors implemented regular gradient descent and approximated the sigmoid function with a least-square polynomial. More importantly, in order to obtain acceptable performances, a vertical packing [11] of the training set was used (and therefore, the whole training algorithm had been vectorized).

Kim et al. [86] proposed a very similar solution to the one described just above [2]. However, they designed a more elaborated packing technique that works row-wise and column-wise. They managed to encode an entire dataset of 1579 samples and 18 features into a single ciphertext. This very aggressive packing method had a significant impact on the efficiency of their protocol, allowing them to train a logistic model over the Edinburgh [87] dataset in 3.6 minutes (versus 114 minutes in [2]).

Both solutions presented above suffer from an important lack of scalability. In fact, by avoiding bootstrapping, the number of gradient descent iterations must remain small (and is pre-computed before the beginning of the training). Otherwise, the parameters of the cryptosystem would grow extremely fast and make the computations dramatically slow. Therefore, these solutions can only work with relatively small datasets.

In order to handle larger datasets, Han et al. [88] designed a very similar solution to [86] but augmented the protocol with the *bootstrapping procedure*. By doing so, they managed to train a logistic model over large datasets. Of course, adding bootstrapping made their protocol significantly slower than [86] for small training sets, but the benefit becomes evident when considering larger inputs. Concretely, using 200 iterations of gradient descent, they managed to fit a dataset of 422,108 samples over 200 features in 1,060 minutes.

Homomorphic Neural Networks Inference (Cryptonets)

Gilad et al. [89] present a system called CryptoNets, which allows homomorphically encrypted data feedforwarding an already-trained neural network. In their setting, a client resorts to a cloud service to perform classification over its private data using a neural

[10] Strictly speaking, the scheme was used as a Somewhat Homomorphic cryptosystem since authors never applied bootstrapping.

[11] Each ciphertext contains all the values of a unique attribute for the whole training set (a column).

network. Since CryptoNets considers that neural network weights are pre-trained, the system aims at predicting individual data items and does not handle the training phase.

Gilad et al. were the first to propose an efficient solution for homomorphic neural network inference. They used the leveled homomorphic encryption scheme YASHE [90] and evaluated their solution on image classification using a small Convolutional Neural Network (CNN) [91]. Authors addressed several challenges to make CNN's more *homomorphically friendly*: (1) they proposed to use a square activation function $f(x) = x^2$ instead of the usual sigmoid or ReLU; (2) they replaced max-pooling layers with mean pooling; (3) they heavily relied on packing to amortize the computation cost of homomorphic encryption.

One should note that, although Cryptonets performed relatively well on small CNNs (59000 predictions per hour when fully exploiting packing), the proposed approach does not scale linearly with the depth of the neural network. The reason is that, for obvious efficiency considerations, Gilad et al. avoided the costly bootstrapping operation, meaning that they never reset the noise level in the ciphers. Consequently, the depth of the neural net needs to be fixed beforehand, and adding more layers requires the selection of larger parameters for the cryptosystem, thereby causing a much higher computation cost.

Hesamifard et al. [92] proposed Cryptodl, an extension of cryptonets. In this work, the authors focused on the approximation of activation functions using low-degree polynomials. They highlighted the fact that using the square function as suggested in Cyptonets is not a viable solution. Hesamifard et al. tried several approximation techniques, such as Taylor expansion, Chebyshev polynomials, and a novel approach based on the derivative of the ReLU function. Their experimental results showed that putting more effort into providing a good activation function positively impacts the accuracy with acceptable computational overhead.

Bourse et al. [93] pointed out two critical limitations of Cryptonets: (1) the lack of scalability due to Somewhat Homomorphic Encryption; (2) the heavy use of packing, i.e., the amortized response time of Cryptonets is good as long as the client has many batched requests. To address these issues, Bourse et al. proposed a scale-invariant approach based on *Fully Homomorphic Encryption (FHE)*. In their proposal, each neuron's output is refreshed through bootstrapping, resulting in arbitrarily deep networks being homomorphically evaluated. They used the scheme proposed by Chillotti et al. [94], known as *TFHE* (Fast Fully Homomorphic Encryption over the Torus), because of its fast bootstrapping operation (but does not support batching). In order to accommodate the supported operations in TFHE, Bourse et al. implemented Binarized Neural Network (BNN), which are neural networks where signals and weights are restricted to the set $\{-1, 1\}$. Overall, this approach led to an efficient solution that outperforms Cryptonets for individual queries. However, it does not match the high throughput of Cryptonets when batching is fully used for small CNNs. Still, the benefit of this approach becomes evident when dealing with deeper networks.

Privacy Preserving Distributed Deep Learning

Shokri et al. [95] proposed a PPML method to train and evaluate DNN. In their work, every participant trains his local dataset with the same neural network model and uses the selective parameter sharing of the model as a technique to benefit from other participants' models without explicit sharing of training inputs.

Shokri et al. relied on a parameter server architecture, where each client sends a fraction of his gradients (1% to 10%) to the server at each iteration to minimize the leakage from the gradients. The authors also proposed to use differential privacy to mitigate the leakage furthermore.

Aono et al. [96] showed that minimizing the number of gradients is not a sufficient protection, even when only 1% of them are shared. Indeed, the authors could partially reconstruct the dataset with a very small fraction of the gradients. Therefore, to mitigate this leakage, Aono et al. proposed encrypting the gradients using additively homomorphic encryption. They instantiated their design using two cryptosystems, namely Paillier and an LWE-based scheme. Their approach incurred an acceptable overhead. Specifically, they reported a running time of 2.5 hours to train a Neural Net over the MNIST dataset and roughly 2-3 higher bandwidth usage.

II.1.5.2 Secure Multi-Party-Based PPML Methods

This section introduces some of the most relevant works in Privacy Preserving Machine Learning that entirely rely on Secure Multi-Party Computation primitives. These solutions are typically less computationally intensive than the ones based on Homomorphic Encryption (HE), but, on the flip side, they usually require a lot of communication and synchronization between the participants and impose a weaker security model.

SecureML

Mohassel and Zhang proposed a two-server model for privacy preserving training [97] in an outsourced setting. Their protocols rely on Secure Multiparty Computation (MPC), specifically Additive Secret Sharing and Garbled Circuits.

Unlike homomorphic encryption-based solutions, MPC protocols require many parties and cannot be deployed on a unique server. These parties are generally the data owners, who jointly build a model on the union of their inputs. However, this configuration differs from the outsourced scenario targeted by SecureML, where clients want to delegate the computation to an external ML server provider (cloud) and remain offline during the whole process. Mohassel and Zhang resolved this issue by using a two-server model as a proxy. Specifically, data owners secret-share their inputs among two *non-colluding* servers to train various models using secure two-party computation (2PC). While their focus is on training, they also support privacy-preserving predictions.

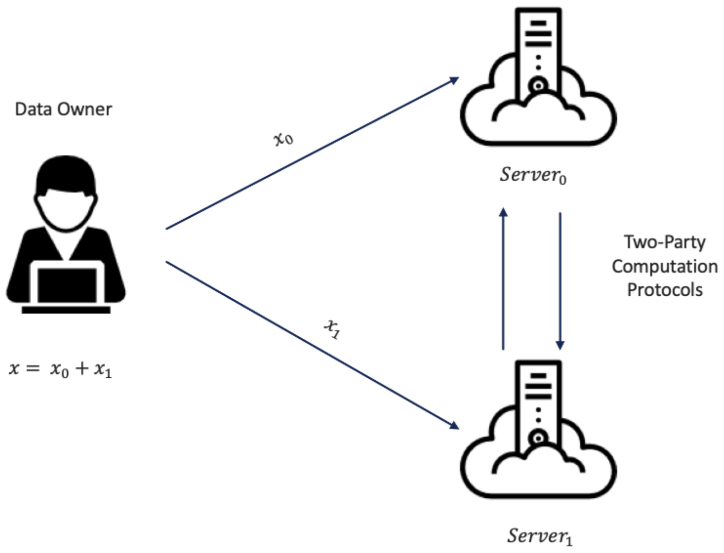


Figure II.1.6: SecureML [97] architecture to achieve Outsourced Privacy Preserving Machine Learning.

Mohassel and Zhang implemented three ML models, namely linear regression, logistic regression, and neural networks. They reported impressive execution times, with only a few seconds (resp. few minutes) to train a linear (resp. logistic) regression over a million samples of hundred features each. On the other hand, training a neural net is significantly more expensive and takes a few hours to complete, even with a single hidden layer.

SecureML introduced many important optimizations. For instance, computations with secret sharing were vectorized in a way that significantly reduced the communication overhead. Moreover, their protocols are divided into an offline and an online phase, which is an interesting paradigm that allows shifting the heavy computations to pre-processing step where the cloud is not solicited. In order to maximize efficiency, authors mixed secret sharing and garbled circuits along with conversion protocols and claimed that they selected the most appropriate protocol for each computation.

Privacy Preserving Decision Trees

Lindell et al. [29] addressed a scenario involving two parties, each one of them holding a dataset of different transactions. The parties wish to compute a decision tree by applying the ID3 algorithm [98] to the union of their databases.

A key observation is that each tree node can be computed separately, with the output made public, before continuing to the next node. In general, private protocols have the property that intermediate values remain hidden. However, in ID3, some of these intermediate values are part of the output and may, therefore, be revealed. Once the attribute of a given node has been found, both parties can separately partition their remaining transactions accordingly for the coming recursive calls. This means that private distributed ID3 can be reduced to privately finding the attribute with the highest information gain.

With an intelligent analysis of the problem, Lindell et al. reduced the problem of securely computing ID3 to the computation of $xln(x)$ using secret sharing and then performing an $argmax$ with garbled circuits.

SecureNN: Privacy Preserving Neural Network Training

Wagh et al. [99] propose SecureNN, a three-party secure computation solution for various Neural Network building blocks such as matrix multiplication, convolutions, Rectified Linear Units (ReLU), Maxpool, normalization, and so on. Using these building blocks, authors managed to construct secure three-party protocols for training and inference of several neural network architectures such that no single party learns any information about the data. The proposed solution outperformed most existing works by at least one order of magnitude. Their main contribution is new and efficient protocols for non-linear functions such as ReLU and Maxpool that avoid using garbled circuits altogether, which constitute the main bottleneck in previous solutions.

Privacy Preservation in Machine Learning Frameworks

Few recent attempts have been made to integrate privacy protection mechanisms into Machine Learning frameworks such as Tensorflow [100], and Pytorch [101]. The main motivations behind these works can be summarized as follows: (1) designing a PPML solution from scratch requires deep knowledge in Machine Learning, Cryptography, and Distributed Systems; (2) most of the implemented solutions remain research prototypes that are never maintained nor re-used. Therefore, it could be highly beneficial to deploy PPML solutions on existing ML platforms to give the user a higher level of abstraction and benefit from the communities of these frameworks.

Dahl et al. [102] developed a library on top of Tensorflow they called *tf-encrypted*. They implemented the well-known SPDZ protocol proposed by Damgard et al. [103], which is a secure multi-party computation protocol based on secret sharing. They augmented it with mechanisms that provide robustness against malicious adversaries. Dahl et al. implemented the protocol in a two-party setting. By translating the protocol into regular Tensorflow code, they benefited from the automatic optimizations of computation graphs built on the framework. Moreover, they did not need to handle the network communication and synchronization between the different parties since Tensorflow also manages this aspect.

Ryffel et al. [104] did similar work on the Deep Learning framework Pytorch and developed a library called PySyft. They implemented the same secure computation protocol (SPDZ) but in a different way. They heavily relied on the concept of *remote-tensor*, which is a tensor located on a remote machine. Moreover, PySyft puts a focus on a federated learning scenario [105], which is quite different from the centralized setting.

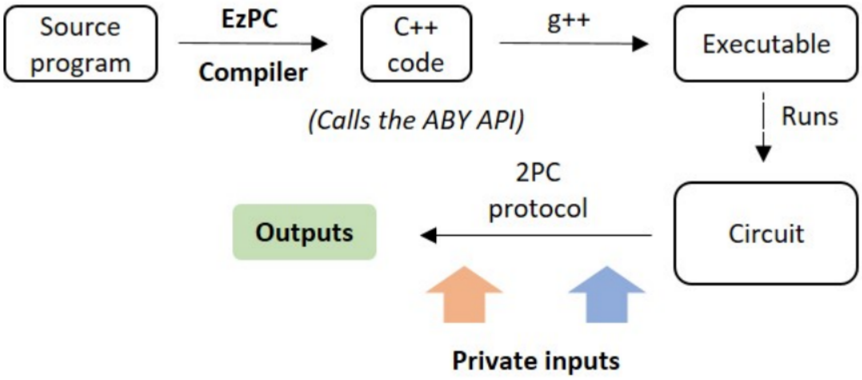


Figure II.1.7: EzPC [106] internal architecture.

Secure Privacy Preserving Compilers

Chandran et al. proposed a new framework for compiling two-party protocols called EzPC [106]. EzPC uses a specialized library called ABY [107] as its cryptographic backend. A simple and easy-to-use imperative programming language is compiled to ABY input, as depicted in Figure II.1.7. An interesting feature of EzPC is its "cost awareness", i.e., its ability to automatically insert type conversion operations to minimize the resulting protocol's total cost. Underneath the hood, secure computation is performed in a two-party setting using a mix of Boolean Sharing, Arithmetic Sharing, and Garbled Circuits. EzPC introduced some mechanisms to choose which protocols should be used at each step of the computation in a cost-aware manner.

The primary contribution of Chandran is a high-level language of secure computation for the non-specialist. Although the resulting program will not be as efficient as a tailored solution designed by an expert, the main gain comes from reducing the development time and a better accessibility of MPC.

It is also worth noting that EzPC is not the first secure computation compiler. However, previous implementations such as CBMC-GC [108], Fairplay [109], Sharemind [110], ObliVM [111], SMCL [112], and Wysteria [113] either rely on secret sharing or garbled circuits, but never combined. Moreover, EzPC was specifically optimized for Machine Learning workflows and is, therefore, more aligned with the scope of this manuscript.

II.1.5.3 Hybrid PPML Methods

In the following, we briefly review the most important and recent hybrid Privacy Preserving Machine Learning works in the literature. These solutions take the approach of combining Homomorphic Encryption schemes with Secure Multi-Party Computation primitives. The motivation behind this is to take advantage of each technique's features and compensate for their respective weaknesses.

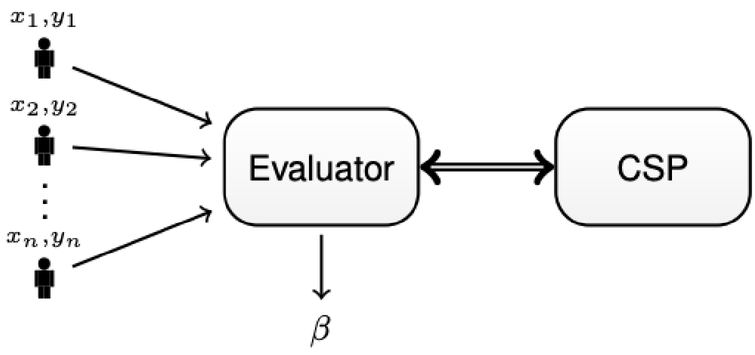


Figure II.1.8: Architecture of the solution proposed by Nikolaenko et. al [114].

Privacy Preserving Ridge Regression

Nikolaenko et al. [114] present a privacy preserving linear regression protocol on horizontally partitioned data using a combination of Additively Homomorphic Encryption (AHE) and Garbled Circuits and evaluate it on datasets with millions of samples. Authors adopted a resolution based on the closed-form of linear regression [115]. The use of garbled circuits imposes a two-party setting (Garbler-Evaluator), where the *Garbler* is responsible for the circuit generation, and the *Evaluator* evaluates the circuit on new inputs. We refer to the two parties as "Evaluator" and "Cloud service provider" (CSP) as denoted in the original paper. They decomposed the training stage of their protocol into 2 phases:

- *Phase 1* clients send outer-products of their feature vectors $a_i = x_i x_i^T$ in encrypted form (using Paillier cryptosystem), as well as a pre-processed right-hand side $b_i = y_i x_i$. The evaluator then sums-up the inputs homomorphically to form the matrix $A = \sum_{i=1}^{n} a_i + \lambda I$ and $b = \sum_{i=1}^{n} b_i$.
- *Phase 2* the evaluator solves the system $Ax = b$ using a garbled circuit that implements Choleskey decomposition. This circuit is pre-computed in an offline phase with the CSP and only depends on the number of features d.

Nikolaenko et al. proposed a secure protocol against a semi-honest Evaluator and CSP along with a variant that is secure even against a malicious CSP. They managed to fit a linear regression for a dataset of 51,000 records and 22 features in roughly 3 minutes. However, their solution is specifically designed for linear regression, and generalizing it to other Machine Learning algorithms is not straightforward.

Secure K-NN

K-NN is a lazy learning classifier, where all computations are deferred until inference time. A data instance is classified based on a majority vote of its neighbors, with the instance being attributed the most common class among its k closest neighbors.

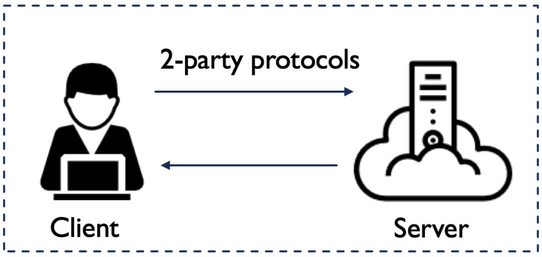


Figure II.1.9: Architecture of the Ciphermed framework proposed by Bost et al. [117].

Kim et al. [116] proposed an approach based on garbled circuits for KNN. They implemented a protocol based on Paillier's cryptosystem to determine the most frequent class. In order to improve the performance of their solution, the authors used an index data structure to extract the k nearest neighbors. The major bottleneck of this work is the generation of the index structure, which must be performed from scratch for each new example added to the training set.

Ciphermed

Bost et al. [117] developed a two-party computation framework and mainly used homomorphic encryption as a privacy preserving technique. This has allowed the authors to implement many classification models, including hyperplane decision, naïve Bayes, and binary decision trees over ciphertexts. The authors used three homomorphic cryptosystems, which are Paillier [47], Goldwasser–Micali [49], and a Somewhat Homomorphic Encryption scheme (YASHE) [90], along with other Secure Multi-Party Computation primitives such as Garbled Circuit. They proposed a client-server architecture where a cloud server responds to clients' classification requests over encrypted data in a privacy preserving fashion. They designed interactive protocols for primitive operations (comparison, arg-max, dot product, etc.) that involve the client during the computation. They also showed how to use these building blocks to construct more complex Machine Learning classifiers, including Viola and Jones face detection algorithm [118]. Notice however, that Bost et al. did not address the training phase, i.e, they supposed a pre-trained plaintext model on the server-side.

Their main contribution is the smart combination of the homomorphic cryptosystems, along with conversion protocols between them. Their protocols were designed to minimize the computational overhead and accept a trade-off on the communication cost. They achieved good performances with execution times ranging from hundreds of milliseconds to a few seconds for a single classification. This work is considered as one of the most important steps towards practical outsourced Privacy Preserving classification using a cloud service.

Privacy Preserving Neural Network Inference

Liu et al. propose MiniONN [119], a privacy preserving neural network prediction framework. Authors designed oblivious protocols for operations routinely used in neural networks: linear transformations, popular activation functions, and pooling operations. In particular, they use polynomial splines to approximate nonlinear functions (e.g., sigmoid and tanh) with negligible loss in prediction accuracy. None of their protocols require any changes to the training phase of the model being transformed. Their protocol is divided into two phases: an independent offline phase and an online prediction phase. On one hand, the online phase only involves lightweight cryptographic primitives such as secret sharing and garbled circuits. On the other hand, the offline pre-computation phase uses additively homomorphic encryption and the SIMD batch processing technique to perform request-independent operations.

Jukevar et al. [120] proposed GAZELLE, the currently most efficient solution for oblivious Convolutional Neural Network (CNN) inference. Their solution relies on a well-designed combination of a Somewhat Homomorphic cryptosystem and garbled circuit. Even though Jukevar et al. relied on a Lattice-based scheme, they never exploited its multiplicative capabilities. The cryptosystem was used as an additively homomorphic one, and the authors claimed that this approach is more efficient than using a classical additive scheme such as Paillier. They introduced numerous optimizations, such as an elaborated packing method and an aggressive parameter selection for their cryptosystem. Their solution outperformed Cryptonets by two to three orders of magnitude in the execution time. Notice however, that their protocol requires an interaction with the client for each activation layer (due to the use of garbled circuits). Nevertheless, Jukevar et al. exploited this constraint to make the client *refresh* the ciphertext, which allowed to control the noise growth due to homomorphic encryption and thereby to gain linear scalability with respect to the depth of the neural network.

II.1.6 Summary of Related Work on PPML

In this chapter, we presented some of the general prerequisites for Privacy Preserving Machine Learning (PPML), including a succinct overview of core Machine Learning concepts, along with basic notions related to privacy. We also introduced the terminology related to the field of Privacy Preserving Machine Learning, and provided a taxonomy of the most important families of PPML methods based on the privacy preservation techniques they use. These two families are cryptography-based and non-cryptographic PPML methods which we recall in Figure II.1.10. They mainly differ in terms of the privacy level they offer, their impact on ML models' utility, and the computational overhead they induce. We presented a general overview of PPML methods in each one of these two families and have shown that none of these PPML methods can outperform all the others with respect to every metric. Specifically, in order to improve one of the aspects, it is necessary to accept a trade-off on the two others.

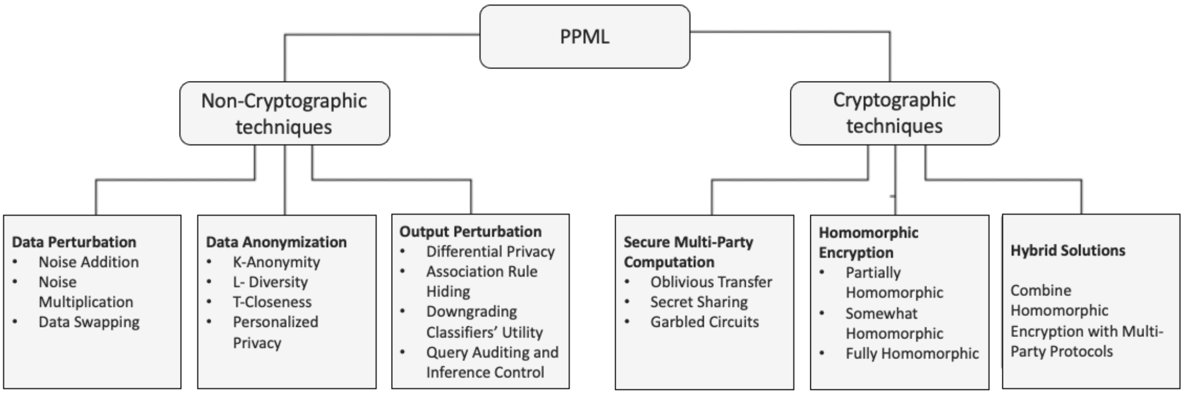


Figure II.1.10: A classification of PPML methods based on privacy preservation mechanisms that they use.

Indeed cryptographic techniques do not alter the accuracy of ML algorithms. However, this utility preservation comes at a price in terms of efficiency. In fact, cryptographic approaches introduce a significant computational/communication overhead and are several orders of magnitude slower than the original ML algorithms. This overhead varies depending on the used cryptographic primitive and is exceptionally high for PPML methods that rely on homomorphic encryption, albeit that these PPML methods employ sophisticated techniques to absorb this overhead, such as ciphertext packing and interesting combinations between various cryptographic primitives that aims to take advantage of each primitive's features and compensate for their respective weaknesses.

On the other hand, non-cryptographic methods rely on altering the underlying data or directly the output of the machine learning model using generalization or perturbation techniques. Accordingly, these methods introduce an explicit trade-off between utility and privacy but are considerably efficient time and space-wise.

Figure II.1.11 illustrates this tradeoff between privacy, utility and runtime between cryptography-based and non-cryptographic PPML methods.

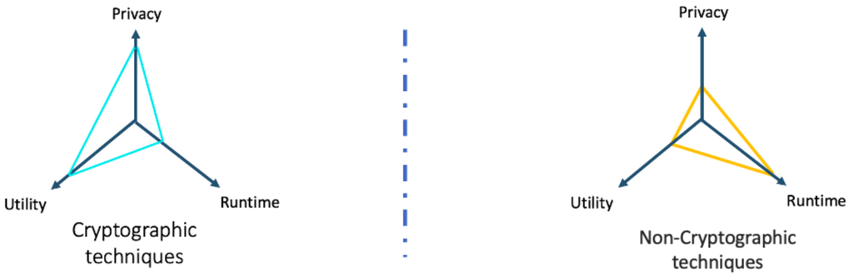


Figure II.1.11: Privacy, Utility, and Runtime tradeoff comparison between cryptographic and non-cryptographic techniques

Finally, cryptography-based approaches provide the strongest and most formal security guarantees, which are usually proven under a mathematical framework and do not rely on any heuristic argument as in non-cryptographic techniques. This has interested us in investigating strategies that enhance the computational efficiency of such PPML methods, specifically those that rely on homomorphic encryption, which are still perceived as impractical despite all the significant performance improvements they have witnessed during the past decade.

II.2 Design Principles of *PrivML*

In this chapter, we present the first contribution of this thesis which falls in the category of cryptography-based privacy preserving machine learning. As shown in earlier parts of this manuscript, many state-of-the-art works propose cryptography-based solutions to ensure privacy preservation in distributed machine learning, specifically using homomorphic encryption. Although these works offer solid, theoretically proven privacy guarantees, they still suffer from impractical overheads time and space-wise. In this long line of work, we propose *PrivML* an outsourced Homomorphic Encryption-based Privacy Preserving Collaborative Machine Learning framework, that allows optimizing runtime and bandwidth consumption for widely used ML algorithms, using many techniques such as ciphertext packing, fast algorithms for large number arithmetic, approximate computations, and parallel computing. In the following, we present the design choices of *PrivML*. We first present the *PrivML* system and threat model. After that, we present the cryptographic primitives that were used in *PrivML* and provide a detailed description of its design principles. Next, we discuss the optimization strategies considered in *PrivML*, as well as a theoretical analysis of the complexity of the building blocks composing it. We finally conclude with security analysis of *PrivML* using a simulation-based approach following the real/ideal world paradigm [121].

II.2.1 System Model and Privacy Requirements

In the following, we describe the global scheme of our outsourced Homomorphic Encryption-based Privacy Preserving Collaborative Machine Learning framework *PrivML*, its privacy requirements, as well as the threat model that it address

II.2.1.1 System Model

The *PrivML* framework consists of an ML service provider ($MLSP$) that interacts with two types of clients: Data Owners (DO) and a set of authorized classification Queriers (Q) as shown in Figure II.2.1.

We assume that a group of l Data Owners $(DO_i)_{i=1}^{l}$ with limited computational power that hold each training data with an identical scheme, would like to use their datasets $D_1, ..., D_l$ to train a global classifier without breaching their individual privacy. Hence, these parties homomorphically encrypt their training datasets and send them to the ML service provider $MLSP$, who builds a global classification model using this data in a privacy preserving manner. The obtained classifier is used by the $MLSP$ to classify encrypted data records sent by queriers Q, without breaching the privacy of neither the received queries nor the responses corresponding to them. When new data samples are available, or other data owners join the system, the $MLSP$ can incorporate this new knowledge in the current model while maintaining the same privacy guarantees. The $MLSP$ is equipped with two units to leverage the services described above: a Master computation Unit (MU) and a Secondary computation Unit (SU). The units MU and SU collaborate in a set of privacy preserving two-party computation protocols (2PC) to perform ML tasks over encrypted data. We assume the existence of an external trusted authority that it is responsible for generating the cryptosystem's global parameters, interacting with the data owners DO and classification queriers Q in order to generate their respective private and public keys, which they use to encrypt and decrypt the data they exchange with the service provider $MLSP$ in a secure manner. It is also responsible for generating the encryption keys used by the computation units MU and SU during both of the learning and prediction phases.

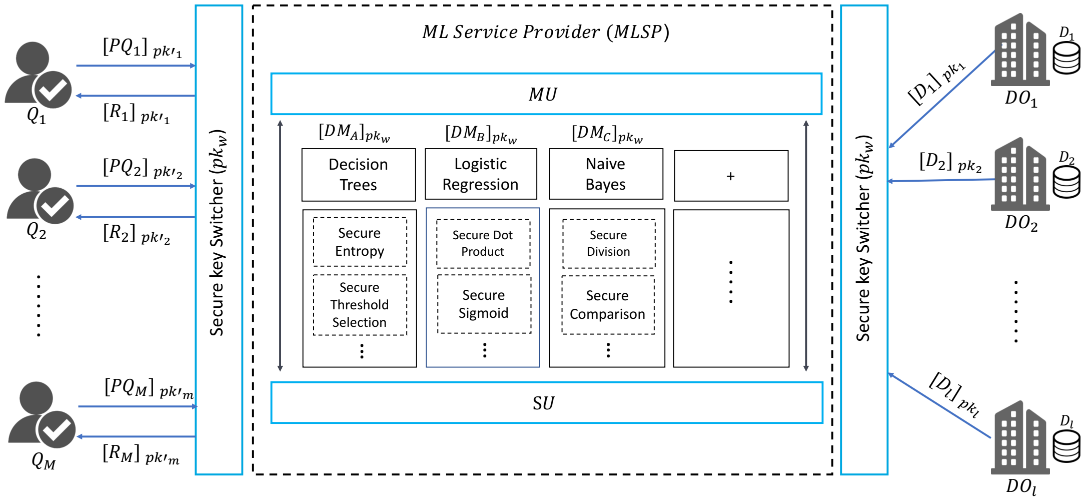


Figure II.2.1: An overview of *PrivML*'s global architecture

II.2.1.2 Threat Model

In our threat model, we consider that data owners $(DO_i)_{i=1}^{l}$, classification queriers Q, and the ML service provider $MLSP$ are semi-honest parties that strictly follow the protocols implemented in *PrivML* but are still curious to infer unauthorized information about private data. We also assume that data owners do not inject any poisonous training data and that the queriers do carry out black-box model inversion attacks, which we consider out of the scope of this work. We introduce an active adversary A^* in our model that aims to get a hold on data records sent to the ML service provider by a challenged data owner DO^* and a classification querier Q^*, as well as classification responses that the $MLSP$ outputs. This adversary may eavesdrop on all communication channels to obtain this encrypted data. A^* is also capable of compromising one or more data owners and queriers except for DO^* and Q^*. It can also compromise at most one of the computation units MU or SU but never both simultaneously.

II.2.1.3 Privacy Requirements

In *PrivML*, we ensure the following privacy preservation guarantees :

1. Data owners $(DO_i)_{i=1}^{l}$ cannot learn each other's individual training data.
2. The ML service provider ($MLSP$) does not learn the exact content of each data owner's training data.
3. The $MLSP$ does not have access to the plaintext parameters of the learned classification model.
4. The $MLSP$ does not get hold of the exact content of the records sent to him by queriers, nor to the classification results corresponding to them.
5. Classification queriers (Q) do not have access to the classification model held by the service provider.

II.2.2 Cryptographic Primitives Underlying *PrivML*

We recall the cryptographic techniques underlying *PrivML*, namely, threshold additive homomorphic encryption and cryptographic blinding.

II.2.2.1 The DT-PKC cryptosystem

In order to entirely outsource the computations required to learn and make use of ML models in *PrivML* to an untrusted machine learning service provider ($MLSP$), we rely on the Distributed Two-Trapdoor Public-Key Cryptosystem (DT-PKC) that was initially proposed by Liu et al. in [82]. The aim of this work was to build a toolkit to run standard operations like multiplication, division, and comparison over data that is encrypted under multiple keys. The DT-PKC cryptosystem has two main properties: the first one is that it is additively homomorphic, which is reflected by Formulas II.2.1a and II.2.1b below, and the second particularity is that it has a distributed double trapdoor decryption mechanism. The idea of double trapdoor decryption was initially proposed by Bresson et al. in [51], where a master key MK can be used to decrypt any cipher that was encrypted under any public key pk generated using the same global parameters as the key MK. In DT-PKC, a similar master key exists but is distributed between two non-colluding parties into two partial strong keys SK_1 and SK_2. In this case, the decryption of a cipher encrypted using

a public key pk, is done in two steps :
$PSdec_1([x]_{pk}, SK_1) = CT$, $PSdec_2([x]_{pk}, CT, SK_2) = x$.

The DT-PKC cryptosystem has the following primitives : $Enc(m, pk) = [m]_{pk}$ to encrypt a message m, $Dec([m]_{pk}, sk)$ to decrypt a cipher $[m]_{pk}$ using the corresponding secret key, $PSdec1([x]_{pk}, SK_1)$, to partially decrypt a cipher by the first party using the first partial strong key SK_1, $PSdec2([x]_{pk}, PSDec_1([x]_{pk}, SK_1), SK_2)$, to entirely decrypt a partial cipher by the second party using SK_2.

II.2.2.2 Cryptographic blinding

As an additively homomorphic cryptosystem, given a plaintext domain Z_N, DT-PKC supports the two following properties:

$$[x]_{pk}.[y]_{pk} = [x + y]_{pk} \tag{II.2.1a}$$

$$[x]_{pk}^{y} = [y.x]_{pk}, y \in Z_N \tag{II.2.1b}$$

These two properties can be used to cryptographically blind operands in a two-party computation protocol (2PC). Assuming that two non-colluding but untrusted parties P_1 and P_2 hold each the cryptosystem's partial strong keys : SK_1 and SK_2 and that these two parties want to compute a function $f\colon {Z_N}^k \to Z_N$ given by $(x_1, ..., x_k) \mapsto f(x_1, ..., x_k)$, where $k \in N$, without knowing neither the inputs $(x_1, ..., x_k)$, nor the output $f(x_1, ..., x_k)$. Given a public key pk generated under the same global parameters as SK_1 and SK_2, the first party receives the encrypted inputs under the same key pk $([x_1]_{pk}, ..., [x_k]_{pk})$, then generates a random value $r = (r_i)_{i=1}^{k} \in Z_N^k$ and uses it to blind the inputs additively using Formula II.2.1a, or multiplicatively using II.2.1b to obtain :
$([B(x_1, r_1)]_{pk} = b_1, ..., [B(x_k, r_k)]_{pk} = b_k)$, where

$$\begin{cases} B\colon {Z_N}^2 \to Z_N \\ (x, \alpha) \mapsto \begin{cases} x + \alpha \\ \alpha.x \end{cases} \end{cases} \tag{II.2.2}$$

After that, this party uses its key SK_1 to partially decrypt the blinded inputs $(b_1, , ..., b_k)$ and sends them to the second party. P_2 uses its part of the strong decryption key SK_2 to entirely decrypt the blinded input via the primitive $PSdec_2$, then applies f over these values to get : $f(B(x_1, r), ..., B(x_k, r))$.
The second party will not be able to access the exact content of $(x_1, ..., x_k)$, nor $f(x_1, ..., x_k)$, since only P_1 knows r. She then re-encrypts this result under pk and sends it back to P_1. Assuming that function f has the analytical properties that allow isolating the actual

input from the blinding value r given a formula of the form :

$$f(B(x_1,r_1),...,B(x_k,r_k)) = \begin{cases} f(x_1,...,x_k) + g(r,x_1,..,x_k) \\ f(x_1,...,x_k).g(r,x_1,..,x_k) \end{cases} \tag{II.2.3}$$

given g: ${Z_N}^{2k} \rightarrow Z_N$, P_1 tries to extract $g(r,x_1,..,x_k)$ by relying Formulas II.2.1a and II.2.1b.

II.2.3 Design Principles of *PrivML*

II.2.3.1 Overview of *PrivML*

In *PrivML*, we use the DT-PKC cryptosystem presented in Section II.2.2, to enable data owners to entirely outsource a machine learning task to the ML service provider $MLSP$, without requiring them to participate in the heavy computations implied by this task. In order to train a classifier over their joint data, Data Owners $(DO_i)_{i=1}^{l}$ dynamically outsource their encrypted data to the ML Service Provider ($MLSP$) in *PrivML*. Considering that mutually untrusted Data Owners must not learn each other's training data, each one of them encrypts his data using a unique public key $(pk_i)_{i=1}^{l}$. In order to use the blinding properties described in Section II.2.2.2, training data and data model parameters are encrypted under the same key. Thus, we use the Secure Key Switching building block described in Protocol 3 to convert all the encrypted training data and prediction queries to a global key pk_w described in (II.2.4), where $\beta \in [1, N/4]$ is an integer randomly generated by the key management infrastructure KMI, while g and N are public parameters of the DT-PKC cryptosystem. $(S_i)_{i=1}^{6}$ are intermediate values computed in each step of this protocol before obtaining its final output. More details about the generation of encryption keys in DT-PKC can be found in [82].

$$\begin{cases} pk_w = g^{\beta}.\prod_{i=1}^{l} pk_i \\ sk_w = \beta + \sum_{i=1}^{l} ski \end{cases} \tag{II.2.4}$$

The global encryption key pk_w is communicated to *PrivML* 's Master computation Unit MU. Not knowing the secret keys held by data owners, this unit cannot construct the global secret key sk_w and, therefore, does not gain access to data encrypted under pk_w.

Training data sent by Data Owners are continuously collected via the Secure Key Switching protocol by the $MLSP$. Periodically (after some time Δt), the $MLSP$ launches a model update process on the newly received training data if a model $[DM_{k-1}]_{pk_w}$, $k > 1$ has been already trained for the considered data scheme. If it is not the case, it is initialized using this data.

At any time, the $MLSP$ can also receive encrypted data records $[PQ]_{pk'_j}$ sent by a prediction Querier Q_j that wants to get the class-label prediction of this record according to the current classification model. The $MLSP$ first changes the encryption key of the query into the global key pk_w, uses the learned data model to get the output $[R]_{pk_w}$ in a privacy preserving manner, then converts it into the Querier's encryption key pk'_j, before sending this response to the Querier.

In the rest of this section, we present a set of building blocks that rely on the 2PC scheme described in Section II.2.2.2. At the basis of these constructions, we design protocols that ensure privacy during both the learning and prediction phases of three classification algorithms, namely Decision Trees, Naive Bayes, and Logistic Regression. The security proofs of these protocols are presented in Section II.2.5.

Protocol 3 Secure Key Switcher

```
    Inputs : $[x]_{pk_a}$, $pk_b$
    Output : $[x]_{pk_b}$
    At $MU$ DO:
    Generate a random number $r \in \mathbb{Z}_N$
    $S_1 \leftarrow [x]_{pk_a}.[r]_{pk_a} = [x+r]_{pk_a}$
    $S_2 \leftarrow PSdec_1(S_1, SK_1)$
    Send $S_1$ and $S_2$ to $SU$
    At $SU$ DO:
    $S_3 \leftarrow PSdec_2(S_1, S_2, SK_2) = x + r$
    $S_4 \leftarrow Enc(S_3, pk_b) = [x+r]_{pk_b}$
    Send $S_4$ to $MU$
    At $MU$ DO:
    $S_5 \leftarrow {[r]_{pk_b}}^{N-1} = [-r]_{pk_b}$
    $S_6 \leftarrow S_5.S_4 = [x+r-r]_{pk_b} = [x]_{pk_b}$
    Return $S_6$
```

II.2.3.2 Privacy Preserving Very Fast Decision Trees

Decision trees consist of a tree-like classification model that is constructed via recursive partitioning of a training dataset into increasingly smaller and more homogeneous subsets of data in terms of target class label. This classifier contains a set of internal nodes, each of which is associated with a test on a given attribute. Each branch incoming from a given node represents the outcome of its classification test. The leaf nodes of the tree correspond to different class labels that are frequently observed there. The selection of test attributes is made based on a specific splitting criterion $G(x)$, where the selected attribute maximizes the homogeneity of leaves that would result if the splitting is done according to this attribute. A Hoeffding tree is a particular type of decision trees that supports online learning. The Very Fast Decision Trees (VFDT) algorithm was proposed in [122] for incrementally training a decision tree in an efficient manner. In this algorithm, a decision tree is gradually updated when new incoming training data is observed. Assuming that the probability distribution of this data does not change over time, the learned tree would be nearly identical to the one that would be constructed if the same training data has been used to train a standard decision tree in batch mode. This is ensured through the

Hoeffding bound H. This bound guarantees, with a high probability of $\theta = 1 - \delta$, that a separation attribute A, selected following the observation of a small number n of training samples at a given node, will still be valid even if a larger number of samples is received, if and only if the difference in terms of the splitting criterion G between this attribute and the second-best attribute B at this level verify equation (II.2.5), where K is the total number of class-labels in the training dataset.

$$\Delta G = G(A) - G(B) > H = \sqrt{log(K)^2 . \frac{ln(1/\delta)}{2n}} \tag{II.2.5}$$

The privacy preserving Very Fast Decision Trees (Priv-VFDT) protocol proposed in *PrivML* allows the ML service provider $MLSP$ to train, dynamically update, and use a decision tree with respect to the privacy requirements defined in Section II.2.1.3. In this protocol, $MLSP$ entirely runs the computations required during both the learning and prediction phases, without having to access neither the content of training data, classification queries, and responses nor the parameters of the learned model. These parameters consist of test attributes, record occurrence statistics, and class labels associated with the decision tree leaves. This protocol results from the composition of the following building blocks in both of the prediction and learning phases : (1) *Secure Comparison,* (2) *Secure Division,* (3) *Secure Multiplication,* (4) *Secure Information Gain computation,* (5) *Secure Threshold Selection, and* (6) *Secure Hoeffding bound evaluation.*

Secure Information Gain Computation

In order to decide whether to split a given leaf L or not after observing a subset S of records in this leaf, and to select the best splitting criterion in case it needs to be split, we use the information gain criterion described in the set of equations (II.2.6). This metric measures the change in information entropy E for the leaf to be split from a prior state to the state where a given classification test T is applied. K is the number of possible classes, $|C_k|$ is the number of records that have the class label C_k in S, p is the number of leaves resulting from the split if the test T is applied (for example, in our case $p = 2$), and $(S_j)_{j=1}^{p}$ are the partitions of records resulting from splitting S into new leaves according to the test T.

$$\begin{cases} G(S,T) = E(S) - E(S|T), \\ E(S) = -\sum_{k=1}^{K}(\frac{|C_k|}{|S|}).log_2(\frac{|C_k|}{|S|}) \\ E(S|T) = \sum_{j=1}^{p}\frac{|S_j|}{|S|}.E(S_j) \end{cases} \tag{II.2.6}$$

To compute information gain over encrypted data, we use the building block described in Protocol 4 to evaluate the entropy E of a given leaf, relying on equation (II.2.7). We also use an adaptation of the Secure Division building block proposed [82] given in Protocol 8.

$$E(S) = log_2(|S|) - \frac{1}{|S|}.\sum_{k=1}^{K} |C_k|.log_2(|C_k|) \tag{II.2.7}$$

Protocol 4 Secure Entropy

```
Input: $([|C_k|]_{pk_w})_{k=1}^{K}, [|S|]_{pk_w}$
Output: $[E(S)]_{pk_w}$
At $MU$ DO:
Generate two random numbers $R_1$ and $R_2 \in \mathbb{Z}_N$
$S_1 \leftarrow ([|S|]_{pk_w})^{R_1} = [R_1.|S|]_{pk_w}$
$S_2 \leftarrow (S_1)^{R_2} = [R_1.R_2.|S|]_{pk_w}$
$S_{3k} \leftarrow ([|C_k|]_{pk_w})^{R_1} = [R_1.|C_k|]_{pk_w},\ k \in 1..K$
$S_4 \leftarrow PSdec_1(S_1, SK_1)$, $S_5 \leftarrow PSdec_1(S_2, SK_1)$,
$S_{6k} \leftarrow PSdec_2(S_{3k}, SK_1)$, $k \in 1..K$
Send $(S_1, S_2, S_{3k}, S_4, S_5, S_{6k}),\ k \in 1..K$ to $SU$
At $SU$ DO:
$S_7 \leftarrow PSdec_2(S_1, S_4, SK_2) = R_1.S$
$S_8 \leftarrow PSdec_2(S_2, S_5, SK_2) = R_1.R_2.S$
$S_{9k} \leftarrow PSdec_2(S_{3k}, S_{6k}, SK_2) = R_1.|C_k|,\ k \in 1..K$
$S_{10} \leftarrow Enc(log_2(S_8) - \frac{1}{S_7}\sum_{k=1}^{K} S_{9k}.log_2(S_{9k}), pk_w)$
    $= [log_2(S) + log_2(R_2) - \frac{1}{S}.\sum_{k=1}^{K}(|C_k|.log_2(|C_k|)]_{pk_w}$
Send $S_{10}$ to $MU$
At $MU$ DO:
$S_{11} \leftarrow [log_2(R_2)]_{pk_w}{}^{N-1} = [-log_2(R_2)]_{pk_b}$
$S_{12} \leftarrow S_{10}.S_{11} = [log_2(S) + log_2(R_2) - log_2(R_2) - \frac{1}{S}.\sum_{k=1}^{K}(|C_k|.log_2(|C_k|)]_{pk_w} = E(S)$
Return $S_{12}$
```

Secure Threshold Selection

In the privacy preserving Very Fast Decision Trees (Priv-VFDT) protocol, we use classification tests of the form $A_i < Th,\ i \in 1..p$, where A_i is one of the data attributes and p is their total number. To select the best splitting test at a given leaf L, we compute the information gain corresponding to every attribute of the dataset using α different potential splitting thresholds $G(S, A_i < Th_{ij}), i \in 1..p, j \in 1..\alpha$, where S is the set of training records observed at L at splitting time.

We use the Gaussian approximation method proposed in [123] to generate these splitting points candidates with little overhead dynamically. This method only requires that the mean and standard deviation of each attribute to be known at the leaf node L subject to splitting. We incrementally update these statistics as new data samples are being observed by computing the sum of values and squares of values of that attribute for each newly observed record. After that, when splitting is to be done, the mean and standard deviation are computed using these running sums.

The splitting thresholds candidates Th_{ij} are given by: $Th_{ij} = \mu_i + \sigma_i.\Phi^{-1}(\frac{j}{\alpha+1})$, where Φ^{-1} is the inverse of the cumulative distribution function of the standard Gaussian, and μ_i and

σ_i are the mean and standard deviation of an attribute A_i at L. Our privacy preserving implementation of the Gaussian threshold selection method is shown in Protocol 5. It relies on the secure division and multiplication building-blocks inspired from [124] and adapted to operands encrypted under the same key. For a given attribute A_i, the sum of this attribute's values $a_{ij}, j \in 1..|S|$ and the sum of their squares are incrementally updated in a privacy preserving manner. *PrivML* 's secure division and multiplication building blocks are presented in Protocols 8, and 9 respectively.

Protocol 5 Secure Threshold Selection

```
  Inputs: $A = [\sum_{j=1}^{|S|} a_{ij}]_{pk_w}$, $B = [\sum_{j=1}^{|S|} a_{ij}^2]_{pk_w}$, $j$, $[|S|]_{pk_w}$, $\lambda$
  Output: $Th_{ij}$
  At $MU$ DO:
  Generate a random numbers $R \in \mathbb{Z}_N$
  $S_1 \leftarrow A^R = [R.\sum_{j=1}^{|S|} a_{ij}]_{pk_w}$
  $S_2 \leftarrow B^{R^2} = [R^2.\sum_{j=1}^{|S|} a_{ij}^2]_{pk_w}$
  $S_3 \leftarrow ([|S|]_{pk_w})^{\sqrt{R}} = [\sqrt{R}.|S|]_{pk_w}$
  $S_4 \leftarrow PSdec_1(S_2, SK_1)$, $S_5 \leftarrow PSdec_1(S_3, SK_1)$,
  $S_6 \leftarrow PSdec_1(S_4, SK_1)$
  Send $S_1$, $S_2$, $S_3$, $S_4$, $S_5$,and $S_6$ to $SU$
  At $SU$ DO:
  $S_7 \leftarrow PSdec_2(S_1, S_4, SK_2) = R.A$
  $S_8 \leftarrow PSdec_2(S_2, S_5, SK_2) = R^2.B$
  $S_9 \leftarrow PSdec_2(S_3, S_6, SK_2) = \sqrt{R}.|S|$
  $S_{10} \leftarrow \Phi^{-1}(\frac{j}{\alpha+1})$
  $S_{11} \leftarrow Enc(S_{10}.\sqrt{\frac{S_8 - S_7^2}{S_9}}, pk_w) = [R^{\frac{3}{4}}.\sigma.\Phi^{-1}(\frac{j}{\alpha+1})]_{pk_w}$
  $S_{12} \leftarrow Enc(\frac{S_7}{S_9}, pk_w) = [\sqrt{R}.\mu]_{pk_w}$
  Send $S_{11}$, $S_{12}$ to $MU$
  At $MU$ DO:
  $S_{13} \leftarrow [R^{\frac{3}{4}}]_{pk_w}$, $S_{14} \leftarrow [\sqrt{R}]_{pk_w}$
  $S_{15} \leftarrow SecureDivision(S_{11}, S_{13}, \lambda)$
  $S_{16} \leftarrow SecureDivision(S_{12}, S_{14}, \lambda)$, $S_{17} \leftarrow S_{15}.S_{16} = [Th_{ij}]_{pk_w}$
  Return $S_{11}$
```

Secure Hoeffding Bound Computation

Given $\delta \in [0, 1]$, the parameter that determines the similarity between batch decision tree and incremental Hoeffding trees learning, and K the number of classes in a training dataset, two publicly known parameters, we use Protocol 6 to evaluate the Hoeffding bound in a privacy preserving manner at a given leaf L subject to splitting. S represents the set of training records observed at L.

Secure Comparison

In order to achieve secure comparison over encrypted operands, we use an adaptation of the building block proposed in [125]. Secure comparison is required during both the learning and prediction phases in the Priv-VFDT protocol, where the comparison operation OP

Protocol 6 Secure Hoeffding Bound Computation

```
    **Inputs:** $log(K)^2$, $ln(1/\delta)$, $[|S|]_{pk_w}$,$\lambda$
    **Output:** $[H]_{pk_w} = [\sqrt{log(K)^2.\frac{ln(1/\delta)}{2.|S|}}]_{pk_w}$
    **At** $MU$ **DO**:
    Generate two random number $R_1$ and $r \in \mathbb{Z}_N$
    $S_1 \leftarrow log(K)^2.ln(1/\delta).r.R_1$, $S_2 \leftarrow ([|S|]_{pk_w})^{2.R_1}$
    $S_3 \leftarrow PSdec_1(S_2, SK_1)$
    Send $S_1$, $S_2$ and $S_3$ to $SU$
    **At** $SU$ **DO**:
    $S_4 \leftarrow PSdec_2(S_2, S_3, SK_2) = 2.|S|.R_1$
    $S_5 \leftarrow Enc(\sqrt{\frac{S_1}{S_4}}, pk_w) = [\sqrt{r}.H]_{pk_w}$
    Send $S_5$ to $MU$
    **At** $MU$ **DO**:
    $S_6 \leftarrow [\sqrt{r}]_{pk_w}$
    $S_7 \leftarrow SecureDivision(S_5, S_6, \lambda)$
    Return $S_7$
```

can be set either to the parameter "is greater than" (>) or "Equal to" (=). In this protocol, the computation units only learn the final result of the comparison without revealing the exact values of the operands. The detailed description of this building block in Protocol 7

Protocol 7 Secure Comparison

```
    **Input :** $[A]_{pk_w}$, $[B]_{pk_w}$, $OP \in \{" > ", " = "\}$
    **Output :** $[A > B]_{pk_w}$ **if** $OP = " > "$ **else** $[A = B]_{pk_w}$
    **At** $MU$ **DO**:
    Generate two random numbers $R_1$ and $R_2 \in Z_N$
    $S_1 \leftarrow ([A]_{pk_w})^{R_1}.[R_2]_{pk_w} = [R_1.A + R_2]_{pk_w}$
    $S_2 \leftarrow ([B]_{pk_w})^{R_1}.[R_2]_{pk_w} = [R_1.B + R_2]_{pk_w}$
    $S_3 \leftarrow PSdec_1(S_1, SK_1)$
    $S_4 \leftarrow PSdec_1(S_2, SK_1)$
    Send $S_1$, $S_2$, $S_3$ and $S_4$ to $SU$
    **At** $SU$ **DO**:
    $S_5 \leftarrow PSdec_2(S_1, S_3, SK_2) = R_1.A + R_2$
    $S_6 \leftarrow PSdec_2(S_2, S_4, SK_2) = R_1.B + R_2$
    if $(OP = " > ")$ $S_7 \leftarrow Enc(S_5 > S_6) =$
            $[R_1.A + R_2 > R_1.B.R_2]_{pk_w}$ else
            $S_7 \leftarrow Enc(S_5 = S_6) = [R_1.A + R_2 = R_1.B.R_2]_{pk_w}$
    Send $S_7$ to $MU$
    **At** $MU$ **DO**:
    Return $S_7$
```

Secure Division and Multiplication

In the following we give the detailed description of the Secure Division and Secure Multiplication protocols. These protocols are very similar to the ones proposed in [82] with proper adjustments to our setting, considering that all computations in *PrivML* are done over values encrypted under a single global key pk_w.

Protocol 8 Secure Division

```
Input : $[A]_{pk_w}$, $[B]_{pk_w}$, $\lambda$
Output : $[10^{\lambda}.\lfloor\frac{A}{B}\rfloor]_{pk_w}$
At $MU$ DO:
Generate two random number $R_1$ and $R_2 \in Z_N$
$S_1 \leftarrow ([A]_{pk_w})^{R_1}.([B]_{pk_w})^{R_1.R_2} = [R_1.A + R_1.R_2.B]_{pk_w}$
$S_2 \leftarrow ([B]_{pk_w})^{R_1} = [R_1.B]_{pk_w}$
$S_3 \leftarrow PSdec_1(S_1, SK_1)$
$S_4 \leftarrow PSdec_1(S_2, SK_1)$
Send $S_1$, $S_2$, $S_3$ and $S_4$ to $SU$
At $SU$ DO:
$S_5 \leftarrow PSdec_2(S_1, S_3, SK_2) = R_1.A + R_1.R_2.B$
$S_6 \leftarrow PSdec_2(S_2, S_4, SK_2) = R_1.B$
$S_7 \leftarrow Enc(10^{\lambda}.\lfloor\frac{S_5}{S_6}\rfloor, pk_w) = [10^{\lambda}.(\lfloor\frac{A}{B}\rfloor + R_2)]_{pk_w}$
Send $S_7$ to $MU$
At $MU$ DO:
$S_8 \leftarrow ([10^{\lambda}.R_2]_{pk_w})^{N-1} = [-10^{\lambda}.R_2]_{pk_w}$
$S_9 \leftarrow S_7.S_8 = [10^{\lambda}.\lfloor\frac{A}{B}\rfloor]_{pk_w}$
Return $S_9$
```

Protocol 9 Secure Multiplication

```
Input : $[A]_{pk_w}$, $[B]_{pk_w}$
Output : $[A.B]_{pk_w}$
At $MU$ DO:
Generate two random numbers $R_1$ and $R_2 \in Z_N$
$S_1 \leftarrow [A]_{pk_w}.[R_1]_{pk_w} = [A + R_1]_{pk_w}$
$S_2 \leftarrow [B]_{pk_w}.[R_2]_{pk_w} = [B + R_2]_{pk_w}$
$S_3 \leftarrow PSdec_1(S_1, SK_1)$
$S_4 \leftarrow PSdec_1(S_2, SK_1)$
Send $S_1$, $S_2$, $S_3$ and $S_4$ to $SU$
At $SU$ DO:
$S_5 \leftarrow PSdec_2(S_1, S_3, SK_2) = A + R_1$
$S_6 \leftarrow PSdec_2(S_2, S_4, SK_2) = B + R_2$
$S_7 \leftarrow Enc(S_5.S_6) = [A.B + R_1.R_2 + R_2.A + R_1.B]_{pk_w}$
Send $S_7$ to $MU$
At $MU$ DO:
$S_8 \leftarrow ([R_1.R_2]_{pk_w}.([A]_{pk_w})^{R_2}.([B]_{pk_w})^{R_1})^{N-1} = [-R_1.R_2 - R_2.A - R_1.B]_{pk_w}$
$S_9 \leftarrow S_7.S_8 = [A.B]_{pk_w}$
Return $S_9$
```

II.2.3.3 Privacy Preserving Naive Bayes

Naive Bayes is a simple yet highly effective classification algorithm. It is used to predict the most probable class label C of a given record $(x_1, .., x_n)$ by applying Bayes's theorem, assuming that every pair of features being classified are independent from each other, as shown in equation (II.2.8).

$$\begin{aligned} c &= argmax_{c_j}(P(c_j|x_1, .., x_n)) \sim \\ c &= argmax_{c_j}(P(c_j)\prod P(a_i|c_j)) \end{aligned} \tag{II.2.8}$$

The prior probabilities $P(c_j)$, and conditional probabilities $P(a_i|c_j)$ are estimated from the training set, typically by computing their respective frequencies. The classification model consists of the conditional and prior class probabilities that are computed from training data. Equations (II.2.9) and (II.2.10) show how these probabilities can be updated when a set of training samples $S = (x_{k1}, x_{k2}, ..., x_{kn}, c_k)_{k=1}^{|S|}$ is available, where θ is the number of training records that had been seen before the update, n_{c_j} is the number of occurrence of the class label c_j in the previously seen data, $|S_{(c=c_j)}|$ is the number of records in S that have the class label c_j, $|S_{((c=c_j)\wedge(a_i=v))}|$ is the number of records that have both a class label c_j and the value v of the attribute a_i, and $|V(a_i)|$ is the number of possible values of this attribute.

For simplicity, we only consider the case of categorical attributes.

$$P(c = c_j) = \frac{\theta.P(c_j) + |S_{(c=c_j)}|}{\theta + |S|} \tag{II.2.9}$$

$$\begin{cases} P(a_i = v|c = c_j) = \frac{\alpha.P(a_i=v|c_j)+|S_{((c=c_j)\wedge(a_i=v))}|}{\alpha+|S_{(c=c_j)}|} \\ \alpha = n_{c_j} + |V(a_i)| \end{cases} \tag{II.2.10}$$

In *PrivML* 's Privacy preserving Naive-Bayes protocol (Priv-NB), we rely on the (1) *Secure Comparison*, (2) *Secure Division*, (3) *and Secure Logarithm* building blocks.

To update and initialize prior class probabilities $P(c_j)$, as well as conditional probabilities $P(a_i|c_j)$ required in this classifier, we rely on a straightforward combination of the Secure Comparison and Secure Logarithm building blocks. Using equations (II.2.9) and (II.2.10) to update these probabilities implies doing one multiplication and one division for each update. Considering that these operations are very costly when done over encrypted data, we keep track of the running sums of occurrences required to compute these probabilities and simply do a single division for each update as shown in equations (II.2.11) and (II.2.12), where n_{c_j} is the number of data samples seen before the update and that have the class label c_j and $n_{((c=c_j)\wedge(a_i=v))}$ is the number of records having both a class label c_j and the value v for the attribute a_i.

$$P(c = c_j) = \frac{n_{c_j} + |S_{(c=c_j)}|}{\theta + |S|} \tag{II.2.11}$$

$$\begin{cases} P(a_i = v|c = c_j) = \frac{n_{((c=c_j)\wedge(a_i=v))}+|S_{((c=c_j)\wedge(a_i=v))}|}{\alpha+|S_{(c=c_j)}|} \\ \alpha = n_{c_j} + |V(a_i)| \end{cases} \tag{II.2.12}$$

The Secure Logarithm and the Secure Probabilities Update protocols are presented in Protocols 10 and 11 respectively. The Secure Probabilities Update protocol takes as inputs two parameters A and B that are set to the nominator and denominator values of the update equations (II.2.11) and (II.2.12) according to the type of the probability to be updated (prior or conditional probability). This protocol outputs both of the updated probability $P = \frac{A}{B}$ and the symmetric value of its logarithm $log(\frac{1}{P}) = -log(P)$.

Moreover, to predict the class label of a given data record $(x_1, .., x_n)$, we transform the classification problem to a minimization problem of the sums of the logarithmic values of these probabilities given in equation (II.2.13).

$$\begin{aligned} c = argmin_{c_j} \sum -log(P(c_j|x_1, .., x_n)) \sim \\ c = argmin_{c_j}(-log(P(c_j)) - \sum log(P(a_i|c_j))) \end{aligned} \tag{II.2.13}$$

Protocol 10 Secure Logarithm

```
    Input : $[A]_{pk_w}$
    Output : $[log(A)]_{pk_w}$
    At $P_1$ DO:
 1: Generate a random numbers $R \in Z_N$
 2: $S_1 \leftarrow ([A]_{pk_w})^R = [R.A]_{pk_w}$
 3: $S_2 \leftarrow PSdec_1(S_1, SK_1)$
 4: Send $S_1$ and $S_2$ to $P_2$
    At $P_2$ DO:
 5: $S_3 \leftarrow PSdec_2(S_1, S_2, SK_2) = R.A$
 6: $S_4 \leftarrow Enc(log(S_3)) = [log(R.A) = log(R) + log(A)]_{pk_w}$
 7: Send $S_4$ to $P_1$
    At $P_1$ DO:
 8: $S_5 \leftarrow ([log(R)]_{pk_w})^{N-1} = [-log(R)]_{pk_w}$
 9: $S_6 \leftarrow S_4.S_5 = [log(A)]_{pk_w}$
10: Return $S_6$
```

Protocol 11 Secure Probabilities Update

```
    Inputs: $[A]_{pk_w}$, $[B]_{pk_w}$, $\lambda$
    Outputs: $[P]_{pk_w} = [\lfloor \frac{A}{B} \rfloor]_{pk_w}$, $[-log(P)]_{pk_w}$
    At $MU$ DO:
 1: $S_1 \leftarrow SecureDivision(A, B, \lambda) = [P]_{pk_w}$
 2: $S_2 \leftarrow SecureLogarithm(S_1, \lambda) = [log(P)]_{pk_w}$
 3: $S_3 \leftarrow S_2^{N-1} = [-log(P)]_{pk_w}$
 4: Return $S_1$, $S_3$
```

II.2.3.4 Privacy Preserving Logistic Regression

Logistic Regression is a statistical method that allows predicting a dependent data variable y by analyzing the relationship between a set of independent variables $(x_1, .., x_n)$. This ML method is used to solve binary classification problems by estimating the probability of a default class C_1 using a Sigmoid function as shown in equation (II.2.14). β is a coefficient vector learned from historical training data.

$$P(Y = 1|X) = \sigma(\beta^T.X) = \frac{1}{1 + e^{-\beta^T.X}} \tag{II.2.14a}$$

$$P(Y = -1|X) = \sigma(-\beta^T.X) = \frac{1}{1 + e^{\beta^T.X}} \tag{II.2.14b}$$

A way to dynamically estimate the logistic model's coefficients is to apply mini-batch stochastic gradient descent (SGD) [126], which is used to incrementally minimize the loss function $\sum_i log(1+e^{-y\beta.X_i})$.

At the observation of a mini-batch set of training data X_B, the update of the coefficients vector using SGD is given in equation II.2.15, where α is the learning rate parameter that controls the convergence speed towards the loss function's minimum and Y_B is the vector of class-labels corresponding to the training data records present in X_B.

$$\beta \leftarrow \beta - \frac{1}{|X_B|}\alpha.X_B^T.(\sigma(X_B.\beta^T) - Y_B) \tag{II.2.15}$$

PrivML 's Privacy preserving Logistic Regression protocol (Priv-LR) relies on two main building blocks: (1) *Secure Sigmoid*, and (2) *Secure Dot Product.* We use the latter to implement privacy preserving matrix multiplication, which is required for the gradient descent update during the learning phase of this classifier.

Secure Sigmoid Evaluation

This building-block described in Protocol 12 allows evaluating the Sigmoid function over $[x]_{pk_w}$, to predict the class label of an encrypted record. Since we can only manipulate integers in the DTPKC cryptosystem, the output returned by this building block is an approximation of the form $\lfloor 10^\lambda.\sigma(u)\rfloor$, where λ stands for the building block's precision. Note that we approximate the sigmoid function with a third-degree polynomial interpolation, that is simpler to implement and less time-consuming.

Protocol 12 Secure Sigmoid

```
    Inputs: $[x]_{pk_w}$, $\lambda$
    Output: $\lfloor 10^\lambda.(0.5 + 1.20096 \cdot (x/8) - 0.81562 \cdot (x/8)^3)\rfloor$
    At $P_1$ DO:
$S_1 \leftarrow [x]_{pk_w}^{10^\lambda * 0.15012}.[10^\lambda * 0.5]_{pk_w} = [10^\lambda * (0.5 + 1.20096 \cdot (x/8))]_{pk_w}$
Generate a random number $R \in \mathbb{Z}_N$
$S_2 \leftarrow [x]_{pk_w}^R = [R.x]_{pk_w}$
$S_3 \leftarrow PSdec_1(S_2, SK_1)$
Send $S_3$ and $S_2$ to $P_2$
    At $P_2$ DO:
$S_4 \leftarrow PSdec_2(S_2, S_3, SK_2) = R.x$
$S_5 \leftarrow Enc(S_4^3) = [R^3.x^3]_{pk_w}$
Send $S_5$ to $P_1$
    At $P_1$ DO:
$S_6 \leftarrow SecureDivision(S_5, [R^3]_{pk_w}, \lambda)$
$S_7 \leftarrow S_6^{-0,0015*10^\lambda}.S_1$
Return $S_7$
```

Secure Dot Product Computation

The building block presented in Protocol 13 is used to compute the dot product of two vectors A and B encrypted under the same encryption key pk_w.

Protocol 13 Secure Dot Product Computation

```
Inputs: $[A = (a_k)_{k=1}^{n}]_{pk_w}$, $[B = (b_k)_{k=1}^{n}]_{pk_w}$, $\lambda$    Output: $A.B$
At $MU$ DO:
    Generate two random numbers $R_1$ and $R_2 \in \mathbb{Z}_N$
    $S_{1k} \leftarrow [a_k]_{pk_w}^{N-R_1}$, $S_{2k} \leftarrow [b_k]_{pk_w}^{N-R_2}, k \in 1..n$
    $S_{3k} \leftarrow PSdec_1(S_{1k}, SK_1)$, $S_{4k} \leftarrow PSdec_1(S_{2k}, SK_1), k \in 1..n$
    Send $S_{1k}, S_{2k}, S_{3k}$ and $S_{4k}$ to $SU$
At $SU$ DO:
    $S_{5k} \leftarrow PSdec_2(S_{1k}, S_{2k}, SK_2) = R_1.a_k,\ k \in 1..n$
    $S_{6k} \leftarrow PSdec_2(S_{3k}, S_{4k}, SK_2) = R_2.b_k,\ k \in 1..n$
    $S_7 \leftarrow Enc(\sum_{k=1}^{n} S_{5k}.S_{6k}) = [R_1.R_2.\sum_{k=1}^{n} a_k.b_k]_{pk_w}$
    Send $S_7$ to $MU$
At $MU$ DO:
    Return $S_8 \leftarrow SecureDivision(S_7, [R_1.R_2]_{pk_w}, \lambda)$
```

II.2.4 Proposed Optimization Techniques

The primary purpose of *PrivML* is to define a methodology for designing end-to-end privacy preserving classification services with practical training and prediction time performance without deteriorating the trained classifiers' utility. To make these design objectives feasible, we rely on the optimization strategies described below:

II.2.4.1 Round Complexity Minimization

A fundamental measure of efficiency in secure Multi-Party Computation protocols is round complexity. Reducing the number of communication rounds between the protocol participants (*PrivML*'s units MU and SU) allows minimizing the effect of network latency and bandwidth, which improves the overall time complexity of the protocol. Therefore, to optimize the performance of the classifiers implemented in *PrivML*, we design building blocks with minimal round complexity instead of the straightforward combination of multiple low-granularity ones as proposed for the DT-PKC cryptosystem [82]. In the following, we consider the same assumptions as in [82] to evaluate the computational complexity of the secure elementary building blocks, where they estimate that an exponentiation operation with an exponent of size $||N||$ requires $1.5||N||$ multiplications. Consequently, the cost of an encryption operation in DT-PKC is in the order of $1.5||N||$, while partial decryption primitive $PSdec_1$ and $PSdec_2$ both have the same cost of $4.5||N||$. They also neglect the cost of a fixed number of additions and multiplications [82]. Considering the computational costs and round complexity of the elementary operations presented in Table II.2.1, we provide in Table II.2.2 a theoretical comparison between the cost of *PrivML*'s optimized secure building blocks and the sequential composition of the elementary secure building blocks. Note that the privacy preserving implementation of elementary operations of exponentiation and square root computation used in this analysis are given at the end of this section in Protocols 15, and 14 respectively. In the case of entropy computation, the Protocol 4 used in *PrivML* is better than the sequential composition of the secure multiplication, division, and logarithm building blocks for a number of classes

$K > 1$, which is always the case in classification problems. The Secure Threshold Selection building block presented in Protocol 5 is better than a sequential elementary-building blocks composition in the case of a number of attributes $p > 5$ of the training dataset.

Protocol 14 Secure Square Root

```
    Input : $[A]_{pk_w}$
    Output : $[\sqrt{A}]_{pk_w}$
    At $P_1$ DO:
    Generate a random numbers $R \in Z_N$
    $S_1 \leftarrow ([A]_{pk_w})^R = [R.A]_{pk_w}$
    $S_2 \leftarrow PSdec_1(S_1, SK_1)$
    Send $S_1$ and $S_2$ to $P_2$
    At $P_2$ DO:
    $S_3 \leftarrow PSdec_2(S_1, S_2, SK_2) = R.A$
    $S_4 \leftarrow Enc(\sqrt{S_3}) = [\sqrt{R}.\sqrt{A}]_{pk_w}$
    Send $S_4$ to $P_1$
    At $P_1$ DO:
    $S_5 \leftarrow [\sqrt{R}]_{pk_w}$
    $S_6 \leftarrow SecureDivision(S_4, S_5)$
    Return $S_6$
```

Protocol 15 Secure Exponentiation

```
    Input : $[A]_{pk_w}$
    Output : $[e^A]_{pk_w}$
    At $MU$ DO:
    Generate a random numbers $R \in Z_N$
    $S_1 \leftarrow [A]_{pk_w}.[R]_{pk_w} = [A + R]_{pk_w}$
    $S_2 \leftarrow PSdec_1(S_1, SK_1)$
    Send $S_1$ and $S_2$ to $SU$
    At $SU$ DO:
    $S_3 \leftarrow PSdec_2(S_1, S_2, SK_2) = R.A$
    $S_4 \leftarrow Enc(e^{S_3}) = [e^R.e^A]_{pk_w}$
    Send $S_4$ to $MU$
    At $MU$ DO:
    $S_5 \leftarrow [e^R]_{pk_w}$
    $S_6 \leftarrow SecureDivision(S_4, S_5)$
    Return $S_6$
```

Building block	Computational cost (#multiplications)	Round complexity
Multiplication	$33\|\|N\|\|$	2
Division	$28.5\|\|N\|\|$	2
Subtraction	$1.5\|\|N\|\|$	0
Comparison	$33\|\|N\|\|$	2
Logarithm	$18\|\|N\|\|$	2
Square root	$45\|\|N\|\|$	2
Exponentiation function	$46.5\|\|N\|\|$	2

Table II.2.1: Computational cost and round complexity of elementary secure building blocks

Operation	Cost of elementary composition of secure building blocks		Cost of *PrivML*'s optimized secure building blocks	
	Computational cost (#multiplications)	Round complexity	Computational cost (#multiplications)	Round complexity
Entropy computation	$(78.5K - 31.5)\|\|N\|\|$	$6K - 2$	$(10.5K + 28.5)\|\|N\|\|$	2
Threshold Selection	$168\|\|N\|\|$	8	$100.5\|\|N\|\|$	10
Hoeffding Bound computation	$108\|\|N\|\|$	6	$45\|\|N\|\|$	4
Naive Bayes Probabilities update	63	4	63	4
Sigmoid function computation	76.5	4	48	4
Dot product computation	$33(p-1)\|\|N\|\|$	$2(p-1)$	$(21p + 31.5)\|\|N\|\|$	4

Table II.2.2: Cost of *PrivML*'s secure building blocks vs. sequential composition of elementary building blocks

II.2.4.2 Logarithmic Probabilities for Naive Bayes

In Naive Bayes classifier, in order to predict the class label of a given data record using equation (II.2.8), Kp and $K - 1$ comparisons are required, where K is the number of class values and p is the number of data attributes. This results in a computational cost of $33(Kp + K - 1)||N||$ and a round complexity of $2Kp + 2K - 2$ rounds. To reduce these costs, we use logarithmic probabilities in the prediction phase of this classifier as discussed in Section II.2.3.3 by applying equation (II.2.13). This reduces the computational cost to $33(K - 1)||N||$ with a round complexity of $2K - 2$ rounds, considering that the cost of a homomorphic addition is insignificant compared to the Secure Multiplication building block.

II.2.4.3 Random Large Numbers and Powers Pre-computation

Random large numbers are used in *PrivML* for generating the noise part of ciphertext or for cryptographic blinding operations. Similar to Beaver Multiplication Triples pre-computation for secret sharing [97], the idea is to carry offline pre-computation that are not data-specific to absorb the computational overhead without influencing the security/privacy guarantees of the system. In the case of the encryption primitive, in DT-PKC to encrypt a plaintext value m under a public key pk_i a random sufficiently big number $r \in [1, N/4]$ is generated and the ciphertext can be obtained as $[m]_{pk_i} = \{T_{i,1}, T_{i,2}\}$, where $T_{i,1} = g^{r\theta_i}(1 + mN) mod N^2$; $T_{i,2} = g^r mod N^2$. We have noticed that the computation of the random number r and its corresponding random power g^r [1] are the most time consuming during encryption and in the same time are not dependant on online operands. This makes it possible to pre-compute such values. Nonetheless, this strategy needs to be applied carefully since the semantic security of the cryptographic primitives underlying *PrivML* depends considerably on the randomness of such values. The idea is to pre-compute many random powers of g^r and use them to produce new random powers. This works because any product of random powers of $g^{r_1} g^{r_2}$ will produce a new random power $g^{r_1+r_2}$. So instead of carrying exponentiation during the online phase of our protocols, we transform it to multiplications of randomly pre-computed powers, which is significantly less consuming. Obviously, there is a trade-off between the storage of pre-computed powers and their "randomness." In order to evaluate the randomness of these values, we follow the guidelines provided in [127] to optimize the efficiency of the Paillier cryptosystem. These guidelines ensure that these pre-computations do not influence the security of the choice of r that needs to be hard to guess by an attacker. To do that, the authors defined the number of pre-computed powers required to obtain a certain security level. For instance, in most of our experiments, we want to ensure an 80-bit security level; therefore, we consider pre-computing a table of 2^{20} random powers and multiplying 5 of them together. The same idea is used for blinding values employed in *PrivML*'s building blocks.

II.2.4.4 Optimized Large Number Arithmetic

The high computational cost associated with cryptography-based PPML methods is closely dependant on the efficiency of the algorithms used to carry large numbers arithmetic. Considering that large number multiplication (which is also at the base of these numbers' exponentiation) is a frequently used operation in the cryptographic primitives underlying *PrivML*, we wisely choose the Schönhage and Strassen FFT fast multiplication algorithm that relies on Fourier transforms, which is one of the best-known algorithms for multiplying large integers [128]. We use a low-level assembly-based sub-routine provided in the GMP library [129] for optimal performance.

II.2.4.5 Parallel Computing

PrivML uses parallel computing to reduce the computational overhead and improve our protocols' scalability when the dataset size grows. In the case of *PrivML*'s Very Fast Decision Tree protocol, when trying to split a given leaf L, we compute information gain simultaneously for different attributes $(A_i)_{i=1}^{p}$ and for different splitting threshold

[1] g is a global parameter of DT-PKC corresponding to a generator of order $\frac{1}{2}(p-1)(q-1)$

$(Th_{ij})_{j=1}^{\alpha}$, since there is no computational dependency between them. The selection of possible thresholds is also made in parallel for multiple splitting attribute candidates.

Similarly, we perform a simultaneous update of occurrence statistics of attribute values and class labels when receiving new training records, in both of *PrivML*'s Very Fast Decision Tree protocol and Naive Bayes protocols. We do the same when updating the conditional and prior probabilities in the Naive Bayes classifier.

II.2.4.6 Incremental Model Learning

Continuous model adaptation based on dynamic data arrival is a prevalent scenario in nowadays online ML services. However, most of the existing privacy preserving machine learning methods restrict to the classical batch learning setting. This can be very disadvantageous, considering the overhead that comes with existing cryptographic privacy preserving techniques. In *PrivML*, we implement privacy preserving online classifier learning algorithms, which allows updating private data models when new data owners join the collaborative learning process or when new training data is available, without having to re-parse all the data handled so far and start the training from scratch.

II.2.5 Security Analysis

In this section, we provide simulation-based security proofs [121] of the privacy preserving classification protocols proposed in *PrivML* according to the threat model and privacy requirements described in Section II.2.1. The security of these protocols mainly relies on the semantic security of the DT-PKC scheme [82] under the assumption of honest-but-curious holders of the partial strong keys SK_1 and SK_2, which is the case of the adversary model considered in *PrivML*.

II.2.5.1 Security of *PrivML*'s Building Blocks

In the following, we provide the proof for the Secure Entropy building block presented in Protocol 4. The other building blocks rely on the exact same paradigm as presented in Section II.2.2.2 and thus, have similar security proofs.

Theorem the building block proposed in Protocol 4 allows to correctly compute the entropy of a given leaf node L according to equation (II.2.7) and is secure against a set of semi-honest adversaries $A = (A_{MU}, A_{SU})$.

Proof let $Sim = \{Sim_{A_{MU}}, Sim_{A_{SU}}\}$ be a set of simulators of semi-honest adversaries $A = (A_{MU}, A_{SU})$. $Sim_{A_{MU}}$ simulates the behavior of adversary A_{MU}. It first generates $K+1$ random fictitious encrypted inputs $([|C_k^*|]_{pk_w})_{k=1}^{K}, [|S^*|]_{pk_w}$ by choosing random values for $(|C_k^*|)_{k=1}^{K}$ *and* $|S^*|_{pk_w}$ respectively, and encrypting them via the encryption key pk_w. Using these random inputs, $Sim_{A_{MU}}$ runs the first round of the Secure Entropy protocol

(Instructions 1-7) which is assigned to the first computation unit MU by generating two random numbers R_1, $R_2 \in \mathbb{Z}_N$, then using them to compute values $S_1, S_2, S_{3k},\ k \in 1..K$ based on the homomorphic properties of the DT-PKC cryptosystem. $Sim_{A_{MU}}$ then computes the partial ciphers $S_4, S_5, S_{6k},\ k \in 1..K$ using the first partial decryption mechanism $PSdec_1$, and sends the outputs $(S_1, S_2, S_{3k}, S_4, S_5, S_{6k}),\ k \in 1..K$ to A_{MU}. If A_{MU} replies with $\perp$, then $Sim_{A_{MU}}$ returns $\perp$ as well. The view of A_{MU} consists of the fictitious encrypted inputs it generates. In both of the real and ideal executions, A_{MU} receives the values $(S_1, S_2, S_{3k}, S_4, S_5, S_{6k}),\ k \in 1..K$. This adversary's real and ideal views are indistinguishable due to the semantic security of the DT-PKC cryptosystem under the semi-honest model. This is also the case of the third round of the Secure Entropy protocol (Instructions 13-15), which is also assigned to MU.

$Sim_{A_{SU}}$ on the other hand, simulates the behavior of adversary A_{SU}, by randomly choosing a fictitious encrypted value for S_{10}^* (Instruction 11) and sends it to A_{SU}. If A_{SU} replies with $\perp$, then $Sim_{A_{SU}}$ returns $\perp$ as well. The view of A_{SU} is limited to the random encrypted value it generates. In both real and ideal executions, A_{SU} gets the value S_{10}^*. Thus, these executions are indistinguishable due to the semantic security of the DT-PKC cryptosystem.

II.2.5.2 Security of *PrivML*'s Classifiers

The security of the classifiers implemented in *PrivML* is based on the modular sequential composition of the secure building blocks they rely on [130]. Moreover, an active adversary A^* that intercepts communication between a target Data Owner DO or a target Querier Q and the service provider $MLSP$ does not learn anything about training data, classification queries and responses due to the semantic security of DT-PKC.

Suppose this adversary corrupts one of the computation units MU or SU, or eavesdrops on the communication channel between them. In that case, it does not learn anything on the intermediary results they exchange or the protected classification model parameters. This is explained by the fact that the values exchanged between these units are either encrypted values under pk_w, partial ciphers decrypted via the primitive $PSdec_1$, or plaintext values blinded with random integers in $\mathbb{Z}_N$. In these three situations, the adversary cannot infer plaintext values without having access to either the strong key SK, the global secret key sk_w, or the random values used for blinding. This adversary has only access to one of the partial strong keys, either SK_1 or SK_2, and therefore cannot reconstruct the strong key SK. Adversary A^* also cannot reconstruct the secret global key sk_w since he does not know the secret keys held by Data Owners. Even if the adversary A^* corrupts the secondary unit SU and has access to blinded plaintext operands, he cannot remove the blinding values, which are only known by the master unit MU.

II.2.6 Summary

This chapter presented the design principles of *PrivML*, an outsourced Homomorphic Encryption-based Privacy Preserving Collaborative Machine Learning framework. We first presented its system and threat model. Then we explained the cryptographic primitives used to build the sub-protocols composing *PrivML*, which are the Distributed Two-Trapdoor Public-Key Cryptosystem (DT-PKC) and cryptographic blinding. After that, we discussed *PrivML*'s design principles, optimization strategies, and a security analysis of its sub-protocols. In the next chapter, we present the evaluation results of *PrivML* using real-world datasets in terms of efficiency time and space-wise and compare it to the closest state-of-the-art PPML methods.

II.3 Evaluation of *PrivML*

In this chapter, we present the empirical evaluation of *PrivML* whose design principles were discussed earlier. We first describe the implementation details of the *PrivML* framework and introduce our hardware and software experimental setup. After that, we present the end-to-end evaluation results of this framework's performance. Then, to get a better insight on these high-level results, we carry a low-level performance evaluation of the cryptographic primitives and building blocks underlying *PrivML* and study the impact of the optimization strategies applied at this level on *PrivML*'s PPML protocols performance. Finally, we carry a comparative analysis with the most relevant state-of-the-art solutions time and space-wise.

II.3.1 Implementation Details of *PrivML*

We implemented the *PrivML* framework as a C++ library that is publicaly available for other researchers and practitioners [1]. It includes the Machine Learning algorithms described in Section III.2, as well as their corresponding non-privacy preserving versions for comparison purposes. *PrivML* uses the OMP 4.5 library [131] for parallel computing. The framework also relies on GMP 6.1.2 [129], and MPFR 4.0.1 [132] libraries to implement the proposed cryptographic building blocks and primitives of the DT-PKC cryptosystem.

Note that we provide our own implementation of the DT-PKC cryptosystem described in [133] as a respond to the correctness issue raised in [134].

Our framework consists of five modules: (1) a *Data and Model Manager*, (2) a *Cryptographic Computation Manager*, (3) a *PPML Engine*, (4) an *ML Engine* and (5) a *Test Bench*. The *Data and Model Manager* handles the received training data, classification queries, and the learned models in their encrypted or plaintext form. The *Cryptographic Computation Manager* is responsible for orchestrating the computations over encrypted data that are assigned to the computation units MU and SU. The *PPML Engine* runs the PPML tasks that consist of initializing ML models, updating them when new training data chunks are available, and using them to respond to incoming private classification queries. The *ML Engine* implements the same tasks over plaintext data. We use the *Test Bench* module to monitor and evaluate the framework's performance. Figure II.3.1 provides an overview of this software architecture and shows the interactions that exist between these components and how they interact with the data owners and classification

[1] https://gitlab.liris.cnrs.fr/rtalbi/DAPPLE-2.0

queriers.

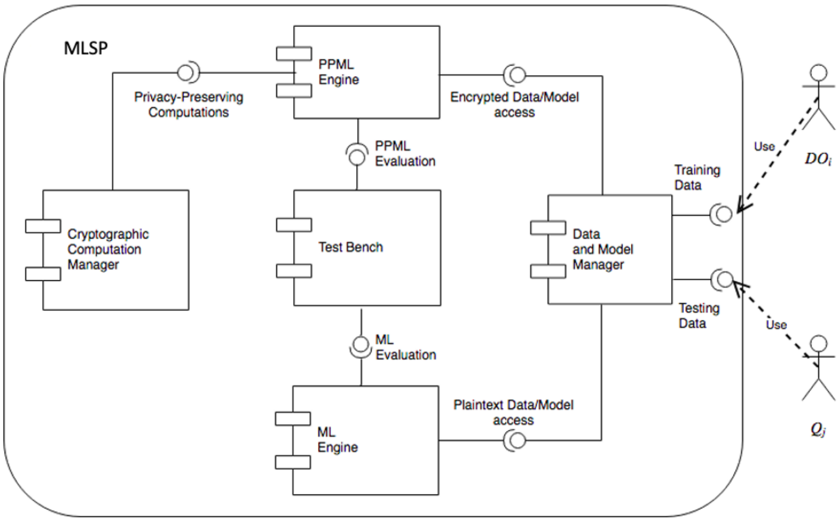


Figure II.3.1: Software architecture of the *PrivML* library

II.3.2 Experimental Setup

II.3.2.1 Hardware Environment

Our experiments were conducted on a Dell PowerEdge C6420 server, with 2 Intel Xeon Gold 6130-Skylake CPUs (2.10 GHz, 16 cores/CPU) and 192 GB RAM. We deployed two KVM virtual machines on this server. The first virtual machine has 12 CPU cores and 16 GB of RAM and stands for the ML Service Provider ($MLSP$), while the second virtual machine has a single CPU core with 4 GB of memory. This machine stands for prediction Queriers Q. For simplicity, in our end-to-end experiments, we assume that the $MLSP$ already holds all the encrypted training data sent by Data Owners in advance. Thus, we do not use an additional virtual machine to emulate interactions with Data Owners.

II.3.2.2 Evaluation Datasets

Our experiments rely on five real-world datasets, which are described in Table II.3.1. The first one is the Adult dataset [135], which represents census data for predicting whether a given individual makes more than $50,000 a year based on attributes, such as education, hours of work per week, etc. The second dataset we use is Bank deposit data [135]. This dataset is related to marketing campaigns of a Portuguese banking institution whose goal was to predict if a client would subscribe to a term deposit or not. The third dataset is the Nursery dataset [135] that aims to rank applications for nursery schools according to social attributes such as parents' occupation and financial state. We also use the Iris dataset [135] for iris plant classification and the Edinburgh dataset for acute myocardial infarction diagnosis in our comparison with state-of-the-art works that are the closest to our setting. All of these datasets are pre-processed by removing missing values and by encoding features into numerical ones.

Dataset	Total dataset size (#records)	#attributes	Training set size (#records)	Testing set size (#records)	#classes
Adult [135]	48 842	14	1 600	400	2
Bank [135]	45 211	16	1 600	400	2
Nursery [135]	129 588	8	1 600	400	5
Iris [135]	150	4	120	30	3
Edinburgh [87]	1 253	9	1 002	251	2

Table II.3.1: Real-world datasets used in *PrivML*'s evaluation

II.3.3 End-to-End Evaluation of *PrivML*

For each of the Adult, Nursery, and Bank deposit datasets [135], we randomly select a subset of 2,000 records while doing proper class re-balancing. We estimated that this would be more practical than using the entire datasets since the experiments we run to evaluate the *PrivML* framework can be very time-consuming for more significant amounts of data.

We split each of these datasets into a training set D containing 80% of the records and a testing set T containing the remaining 20%. The training set is then decomposed into 16 batches $(D_i)_{i=1}^{16}$ of 100 records each to observe the incremental learning of predictive models.

As illustrated in Figure II.3.2, these batches are outsourced to the $MLSP$ by different Data Owners $(DO_i)_{i=1}^{16}$ at different moments. The $MLSP$ learns an initial ML model after receiving the first batch of data which he keeps updating after receiving the remaining data batches. After each update, the current ML model is evaluated using the previously selected testing set.

We have set the following experiments to use an encryption key length of 1024 bits to get an 80-bits security level, with a parallelism level of 14 threads.

II.3.3.1 Evaluation of Private Very Fast Decision Trees

We first evaluate the Priv-VFDT classifier over the Adult, Nursery, and Bank datasets, where we monitor the evolution of training time, prediction time, model accuracy, and complexity. The latter is represented by the number of nodes of the learned decision tree as new training samples are being incrementally outsourced to the ML Service Provider ($MLSP$). We fix the probability $\delta = 10^{-6}$ and the number of possible splitting thresholds

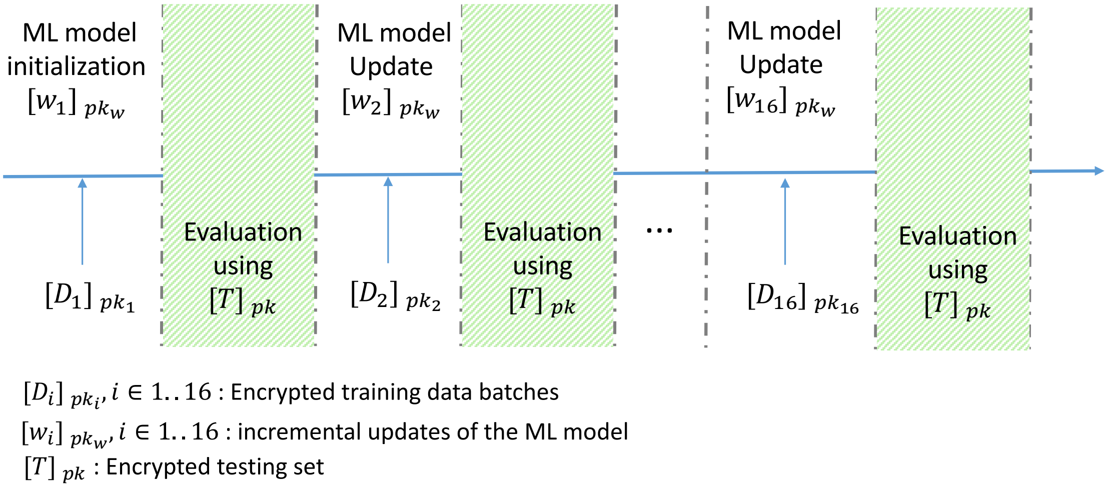
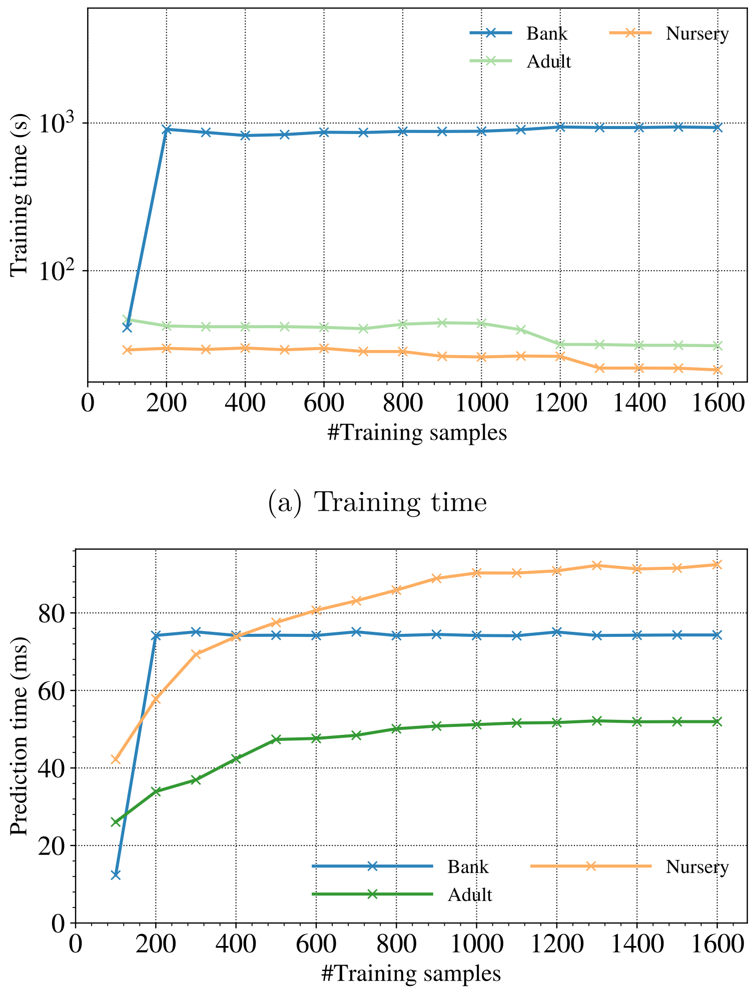

ML model initialization $[w_1]_{pk_w}$
$[D_1]_{pk_1}$
Evaluation using $[T]_{pk}$
ML model Update $[w_2]_{pk_w}$
$[D_2]_{pk_2}$
Evaluation using $[T]_{pk}$
...
ML model Update $[w_{16}]_{pk_w}$
$[D_{16}]_{pk_{16}}$
Evaluation using $[T]_{pk}$
$[D_i]_{pk_i}, i \in 1..16$: Encrypted training data batches
$[w_i]_{pk_w}, i \in 1..16$: incremental updates of the ML model
$[T]_{pk}$: Encrypted testing set

Figure II.3.2: Testing scenario used to evaluate *PrivML*

$\alpha = 2$. We also limit the constructed decision tree growth by setting the maximum number of nodes to 45 in the case of the Adult dataset, to 71 for the Nursery dataset, and 55 for the Bank dataset. Figure II.3.3 illustrates the results obtained by averaging five consecutive runs of the scenario described above for each one of the Adult, Nursery, and Bank datasets using the Priv-VFDT classifier. The entire training process takes an average of 10 minutes for the Adult dataset, 7 minutes for the Nursery dataset, and 3 hours and 43 minutes for the Bank dataset. The training time observed after each model update or initialization is mostly influenced by the number of leaf splits that occur after receiving new training data samples. Once the maximum number of nodes is attained (model update 12 for the Adult and Nursery datasets and update 11 for the Bank dataset), the model reaches a stable state with an average update time of 31 seconds for the Adult dataset, 26 seconds for the Nursery dataset and 13 minutes for the Bank dataset.

As for prediction time, it increases as the number of tree nodes grows. Starting from model update #12 for the Adult and Bank datasets, the learned decision tree reaches the maximum node number. Consequently, prediction time becomes stable with an average value of 74 milliseconds per record for the Bank dataset and 51 milliseconds for the Adult dataset. In the case of the Nursery dataset, the prediction time reaches a stable value of 90 milliseconds once the maximum node number is reached after model update #11. The obtained learning and prediction time obtained using the Priv-VFDT protocol for the three datasets is still extremely high comparing to the plaintext version of this classifier by a factor up to 9000 times. In terms of model utility, the obtained model accuracy is identical to the plaintext version of the VFDT algorithm for both of the Adult and Bank datasets with a maximum value of 71 % and 68 %, respectively. For the Nursery dataset, we obtain a maximum accuracy of 76 % comparing to the plaintext version of the VTDT algorithm that reaches up to 77 % of accuracy. This loss in terms of accuracy is due to the computational approximations done in *PrivML* since the underlaying cryptosystem only supports integer values. Note that the difference in training time between the three



(a) Training time

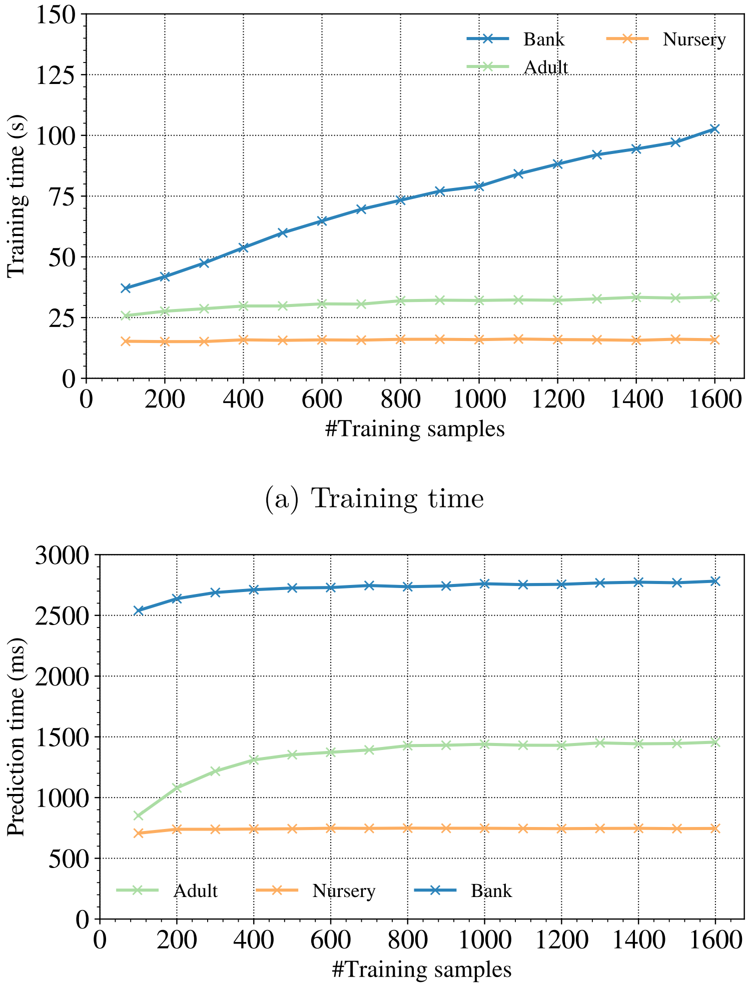
(b) Prediction query execution time

Figure II.3.3: Performance of Private Very Fast Decision Trees

datasets is due to their varying complexity, where the most complex one is the Bank dataset. As for the inference time, it is rather influenced by the learned tree's complexity, where the most complex one belongs to the Nursery dataset.

II.3.3.2 Evaluation of Private Naive Bayes

We also evaluate the Priv-NB protocol over the Adult, Nursery, and Bank datasets in terms of learning time, prediction time, and accuracy. For this classifier, we obtain a practically stable update time for both Adult and Nursery datasets with an average value of 30 and 16 seconds respectively and a total training time of 8 and 4 minutes, respectively. As for the Bank dataset, it has an average update time of 72 seconds with a total learning time of 19 minutes. The update time for this dataset grows as new attribute values are observed for the first time. Thus their corresponding conditional probabilities need to be initialized and updated with the arrival of new data records, which takes additional time. The prediction time for this classifier is stable. It has an average value of 1300, 740, and



(a) Training time

(b) Prediction query execution time

Figure II.3.4: Performance of Private Naive Bayes

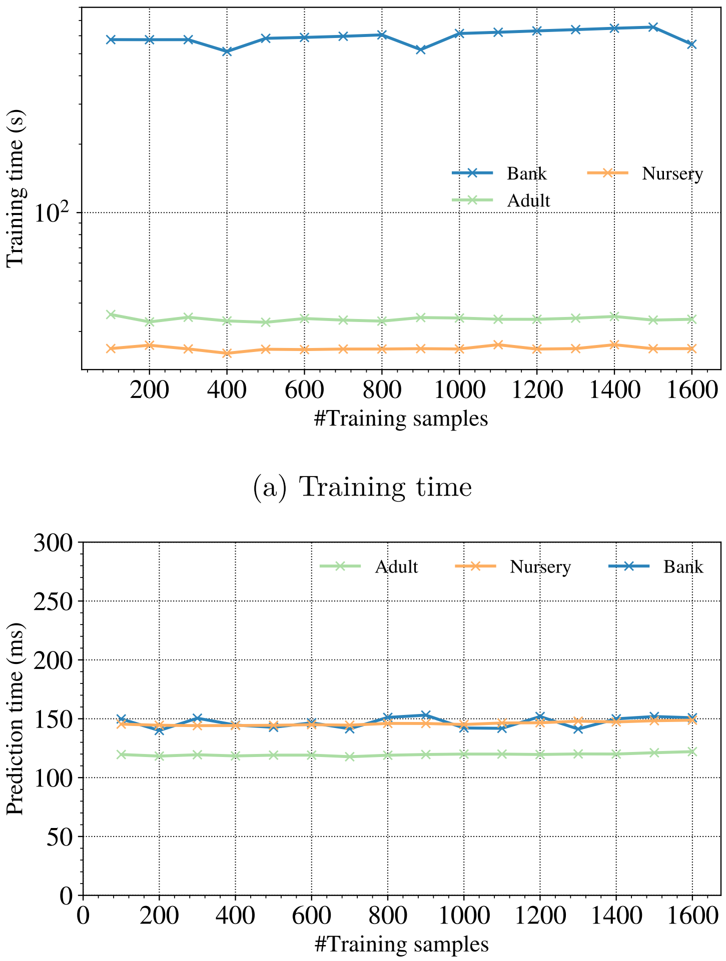
2700 milliseconds for each one of the Adult, Nursery, and Bank datasets, respectively, and grows according to the dataset dimension. We also noticed that using log-probabilities improves prediction time by 20% compared to an implementation where class prediction is performed using equation (II.2.13). In addition, the accuracy obtained using the Priv-NB protocol is identical to the one obtained using the original implementation of this classifier with a maximum value of 77% and 78% for the Nursery and Bank datasets, respectively. For the Adult dataset, a value of 72 % of accuracy is obtained using the Priv-NB protocol compared to a maximum accuracy of 73 % of its plaintext counterpart. The training and prediction time obtained using this classifier are up to 2000 times slower than their plaintext counterparts in terms of computational overhead.

II.3.3.3 Evaluation of Private Logistic Regression

We evaluate the Priv-LR protocol over the Adult, Nursery, and Bank datasets, using a mini-batch size of 100 records and a learning rate $\alpha = 100$ for the Bank and Nursery datasets and a value $\alpha = 1000$ for the Adult dataset. Only one epoch was required for the three datasets to reach a model convergence, where a maximum accuracy of 68 % was obtained for the Adult dataset, 100 % for the Nursery dataset, and 72 % for the Bank dataset.

The model accuracy obtained using the original version of this classifier is identical to its privacy-preserving version implemented in *PrivML* for both of the Adult and Nursery datasets, while it reaches a slightly higher value (73 %) for the Bank dataset.

The update time for this classifier is relatively stable. It has an average value of 9 minutes for the Bank dataset, 34 seconds for the Adult dataset, and 25 seconds for the Nursery dataset. As for prediction time, it is essentially stable with an average value of 146 milliseconds for the Adult and Bank datasets and 122 milliseconds for the Nursery datasets. This steady performance is due to the constant model complexity in logistic regression as the incremental training process progresses, unlike decision trees for instance, whose complexity gradually increases, making training and prediction time increase as well. The learning and prediction time obtained with this classifier is also considerably high comparing to the plaintext counterpart with a value of up to 10,000 times. ‘



(a) Training time

(b) Prediction query execution time

Figure II.3.5: Performance of Private Logistic Regression

II.3.4 Low-Level Evaluation of *PrivML*

To better understand the results obtained in the previous section, we carry a low-level evaluation of *PrivML* where we investigate the impact of our optimization strategies on the performance of cryptographic primitives underlying *PrivML* as well as the building blocks composing its PPML methods.

II.3.4.1 Performance of Underlying Cryptographic Primitives

We first evaluate the impact of our optimization strategies at the lowest granularity level in *PrivML*, where we monitor the efficiency of the cryptographic primitives underlying the DT-PKC cryptosystem. These optimizations are specifically the usage of fast large number arithmetic and random large powers pre-computation.

Figure II.3.6 illustrates the execution time difference between two C++ implementations of these primitives using growing encryption key sizes, where the first one employs our optimizations, while the second one does not. We refer to the latter by naive C++

implementation.

We also include the original version of these primitives provided by the authors who proposed the DT-PKC cryptosystem [133] to illustrate the impact of programming language choice on runtime performance.

In this experiment, we run each one of the three first primitives (Enc, $PSdec_1$, $PSdec_2$) for one million times while we run the primitives SScalarMult and SAdd for at least ten million times since these two exhibit smaller runtimes. These experiments are conducted using a fixed input message size of 48 bits encrypted using 512, 1024, 2048, and 4096 bits encryption keys.

We notice that for the three implementations, the least costly primitives are the homomorphic addition and scalar multiplication that correspond to a large integer multiplication and exponentiation. Using optimized large number arithmetic, the performance of these two primitives is optimized by at least a factor of 2.

The most costly primitives of DT-PKC are the Enc and $PSdec_2$ primitives that correspond to two integer exponentiation and two multiplications in the case of the Enc primitive and to one multiplication and one exponentiation in the case of $PSDec_2$. Using pre-computed random powers, we achieve an improvement up to a factor of 6000 for the Enc primitive and up to a factor of 3 for the $PSdec_2$ primitive.

It is worth mentioning that our optimized C++ implementation of the DT-PKC cryptographic primitives described above is respectively 9000x, 11x, 3x, 21x, and 218x times faster on average than the original Java implementation provided by [133].

II.3.4.2 Performance of Underlying Sub-Protocols

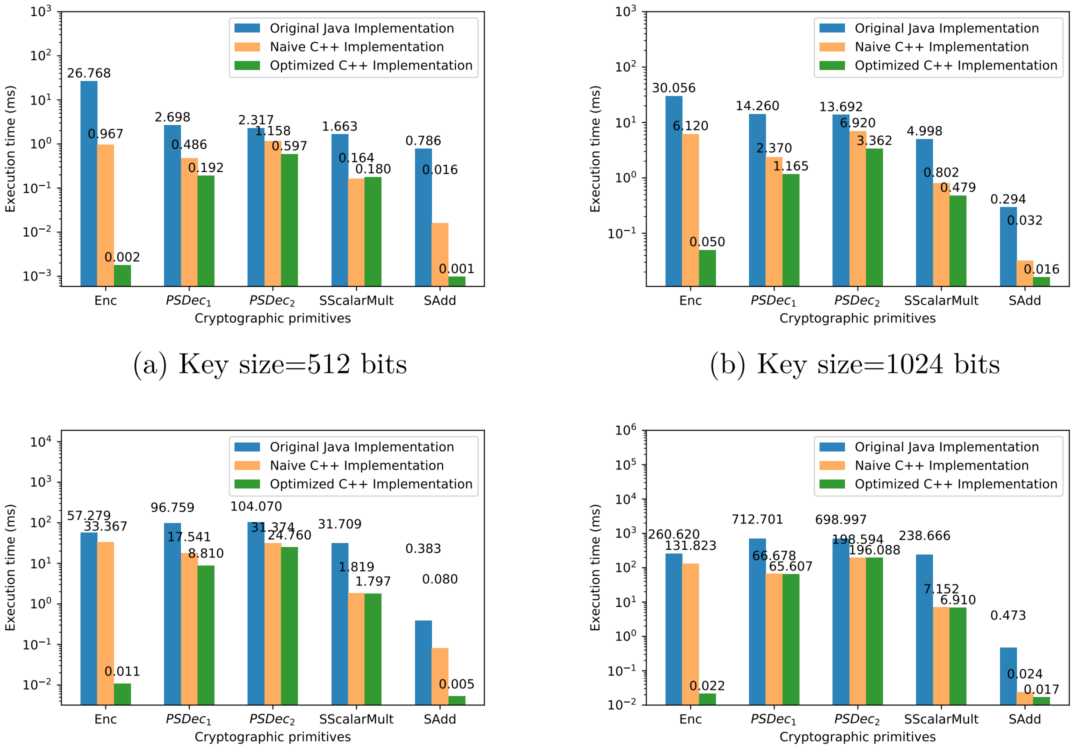
In this experiment, we monitor performance at the granularity of privacy preserving building blocks, also referred to as sub-protocols.

We run each of the sub-protocols proposed in *PrivML* for ten thousand times using a fixed input message size of 48 bits and a 1024 bits encryption key.

We carry a comparison between the time proportion spent in the cryptographic primitives composing each sub-protocol (Enc, $PSdec_1$, $PSdec_2$, $SScalarMult$, $SAdd$), when using their optimized C++ implementation given in Figure II.3.7a and their naive version in Figure II.3.7b. Note that measurements for other non-cryptographic operations such as square root, logarithm, division, etc., are not provided since they exhibit little execution time comparing to the cryptographic primitives.

We notice a significant improvement time-wise when using the optimized primitives to implement *PrivML*'s sub-protocols. At least 9% improvement in runtime is reported in the case of the secure dot product sub-protocol (SDP) and up to 45% improvement in the case of the secure probability update sub-protocol (SPU).

This noticeable improvement is mainly due to the decline of the costs of the Enc and $PSdec_2$ primitives, as demonstrated in the previous section.



(a) Key size=512 bits

(b) Key size=1024 bits

(c) Key size=2048 bits

(d) Key size=4096 bits

Figure II.3.6: Execution time optimization of the primitives underlying DT-PKC with respect to encryption key size

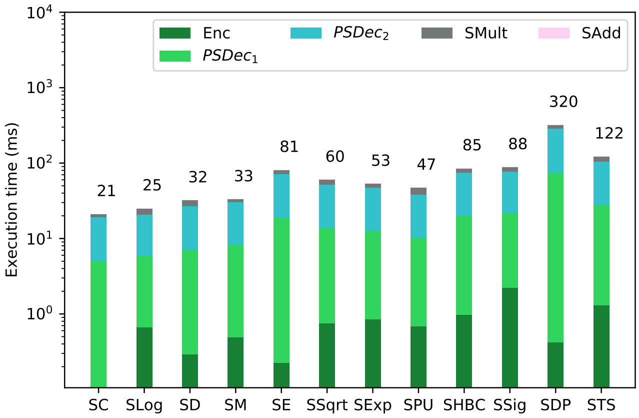
We also notice that the most costly building blocks in *PrivML* are the SDP, SSig, and SE building blocks, respectively, due to their high complexity as theoretically demonstrated in Section II.2.4.1.

We also measure the communication cost in bytes of our sub-protocols which correspond to the amount of data exchanged between the computation units MU and SU (See Figure II.3.7c). In terms of communication cost, the most expensive sub-protocol is the SDP sub-protocol since the manipulated inputs of this sub-protocol which are exchanged after cryptographic blinding between the computation units, are two vectors of a fixed dimension of 14 in this experiment. The least expensive sub-protocol in terms of communication cost is the SC sub-protocol since it has a low round complexity and the exchanged data between the computation units are two blinded scalars in the first round and an encrypted Boolean in the second one.

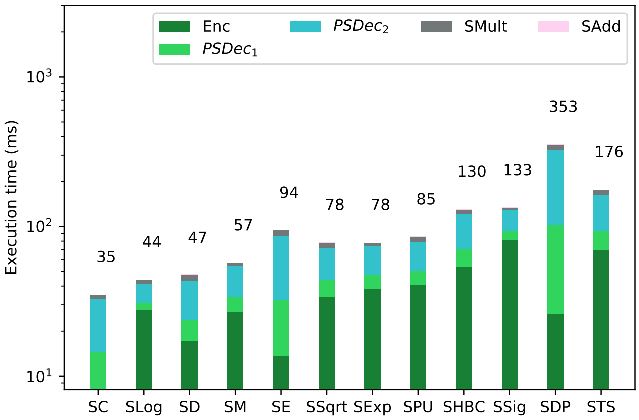
II.3.4.3 End-to-End Microbenchmarks of *PrivML*

Following the efficiency analysis at the cryptographic primitives and sub-protocols granularity in the previous sections, we carry a low-level end-to-end evaluation of *PrivML*'s PPML methods. In analogy to the results presented in section II.3.3, this experiment is carried using the Adult, Nursery, and Bank datasets [135].

In this experiment, we measure the time proportion spent in each sub-protocol imple-



(a) Cryptographic primitives runtime proportion with respect to each sub-protocol execution time in milliseconds using their optimized C++ implementation.



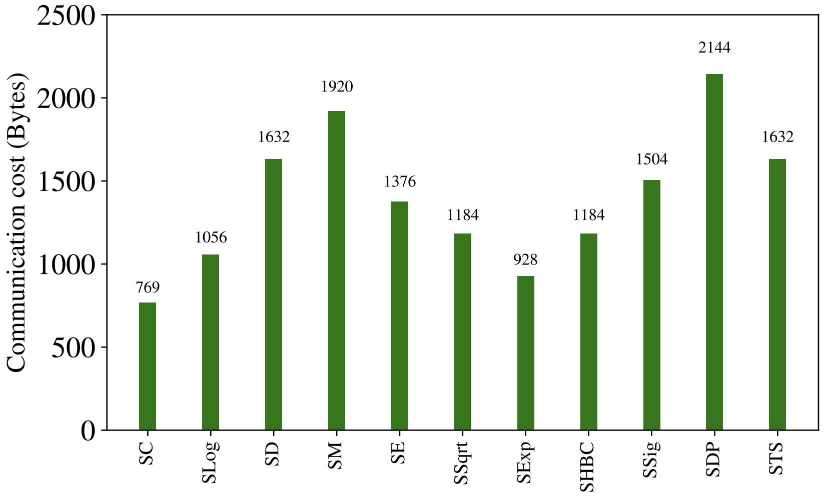
(b) Cryptographic primitives runtime proportion with respect to each sub-protocol execution time in milliseconds using their naive C++ implementation.

mented in *PrivML* as well as the time spent in the elementary cryptographic operations (Enc,$PSdec_1$,$PSdec_2$,$SScalarMult$, $SAdd$) during both training (See Figure II.3.8) and inference phase (See Figure II.3.9).

As shown in Figure II.3.8 the training process is improved by 27%, 19%, 26% on the average for the Priv-NB, the Priv-VFDT, and the Priv-LR PPML methods when employing the optimized C++ primitives' implementation; this is due to performance improvement of highly used sub-protocols such as secure comparison, secure division, and secure multiplication which improved in terms of runtime by 39%, 32%, 41% respectively.

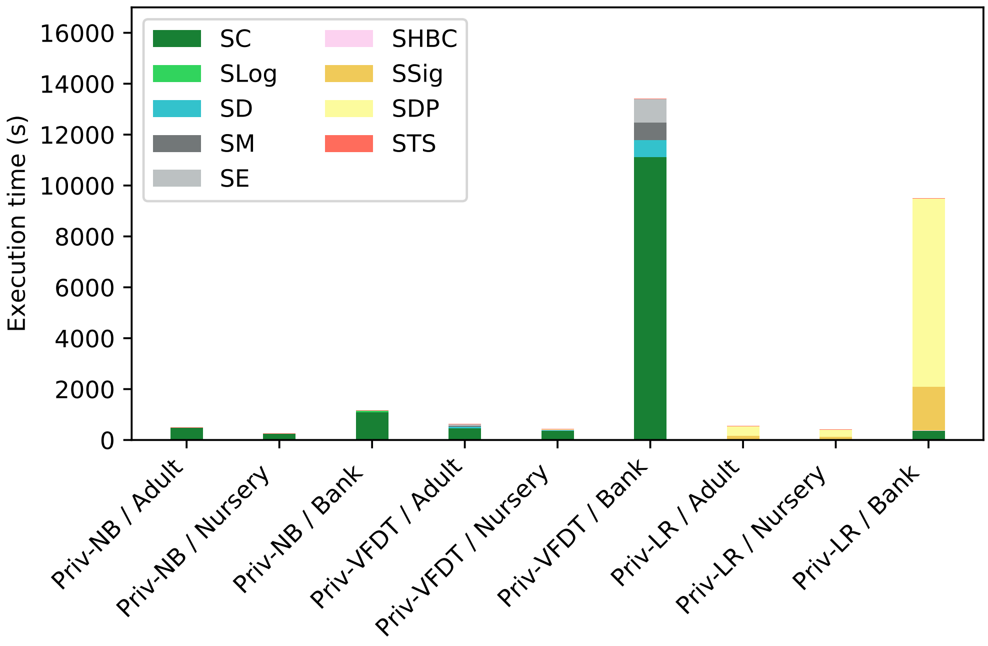
As illustrated in Figure II.3.8-c, this improvement is visible mainly in the absorption of the Enc and $PSDec_2$ primitives costs.

Similar observations can be made during the inference phase as shown in figure II.3.9 where inference time significantly improved by up to 27%, 32%, and 27% for the Priv-NB, the Priv-VFDT, and the Priv-LR PPML methods, respectively.

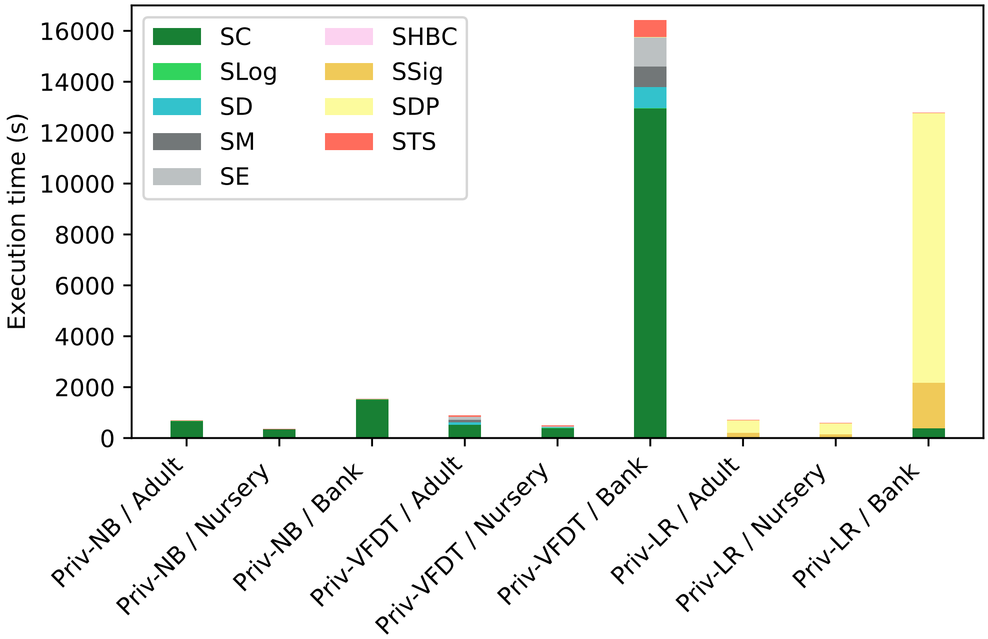


(c) Communication cost in bytes of each sub-protocol in bytes.

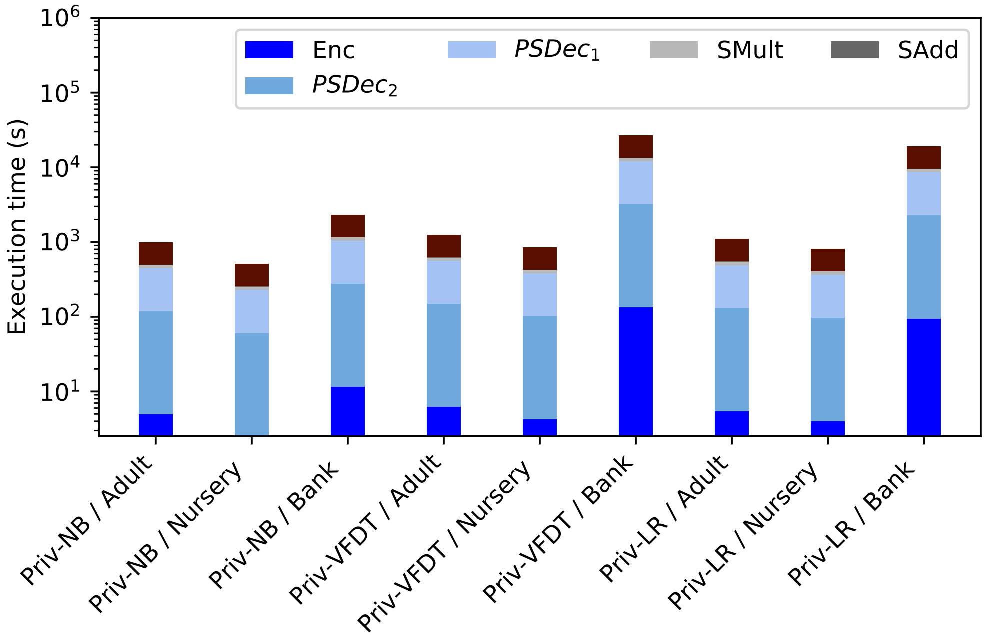
Figure II.3.7: Performance of sub-protocols in terms of execution time and communication cost, SKS : Secure Key Switching, SE : Secure Entropy, STS : Secure Threshold Selection, SHBC : Secure Hoeffding Bound Computation, SSig : Secure Sigmoid computation, SDP : Secure Dot Product, SD : Secure Division, SM : Secure Multiplication, SC : Secure Comparison, SLog : Secure Logarithm, SExpo : Secure Exponentiation, SSqrt : Secure Square Root. Enc: Encryption, $PSdec_1$, $PSdec_2$: Two-step decryption, SScalarMult : Secure Scalar Multiplication, SAdd : Secure Homomorphic Addition. K=5, p=14



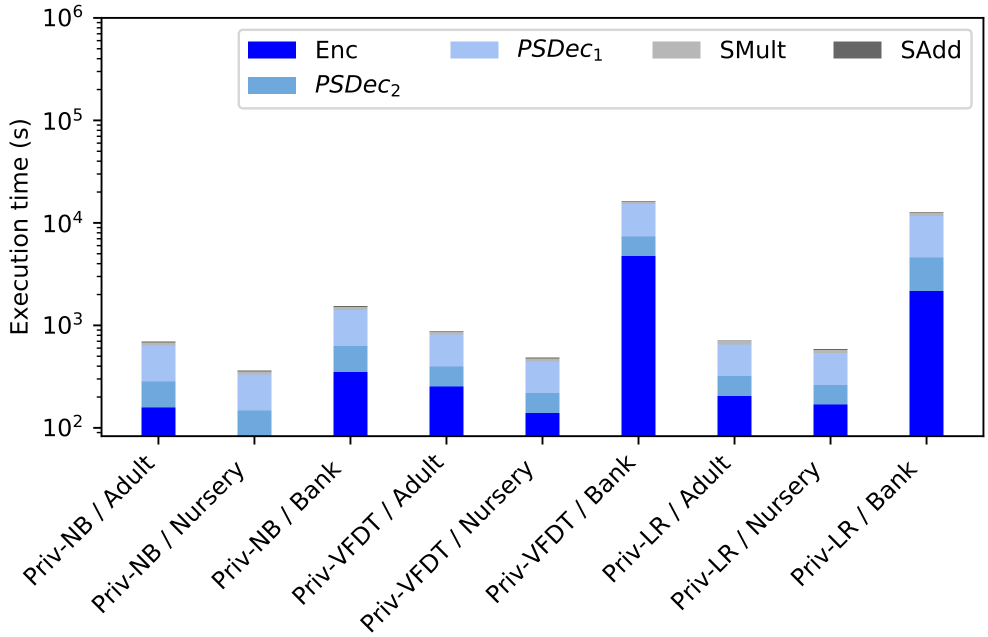
(a) Sub-protocols runtime proportion with respect to each PPML execution time in seconds for a specific dataset (Adult, Nursery, Bank) using optimized C++ implementation.



(b) Sub-protocols runtime proportion with respect to each PPML training time in seconds for a specific dataset (Adult, Nursery, Bank) using naive C++ implementation.

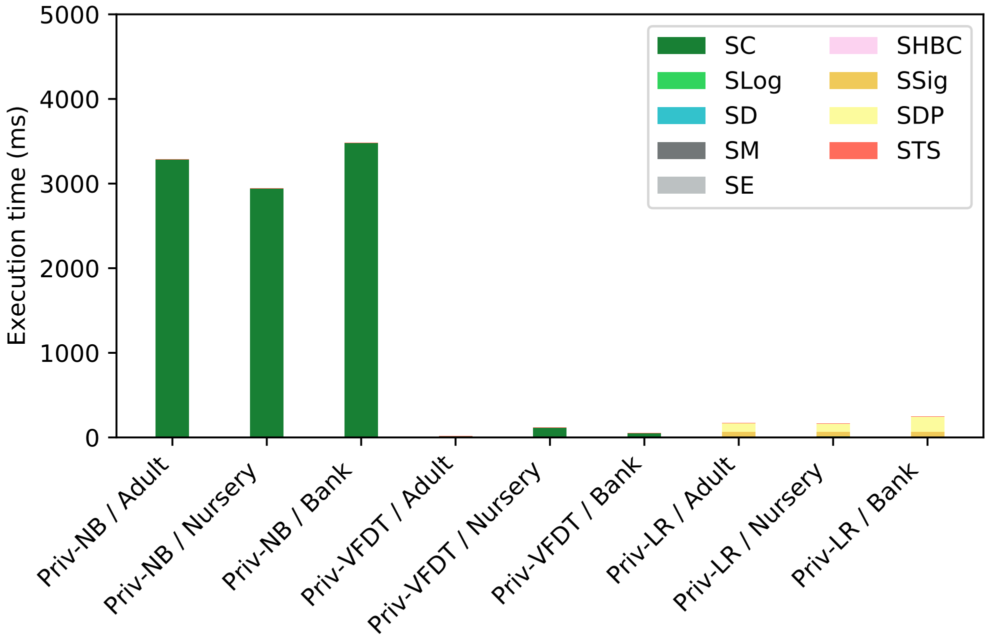


(c) Cryptographic primitives runtime proportion with respect to each PPML training time in seconds for a specific dataset (Adult, Nursery, Bank) using optimized C++ implementation.

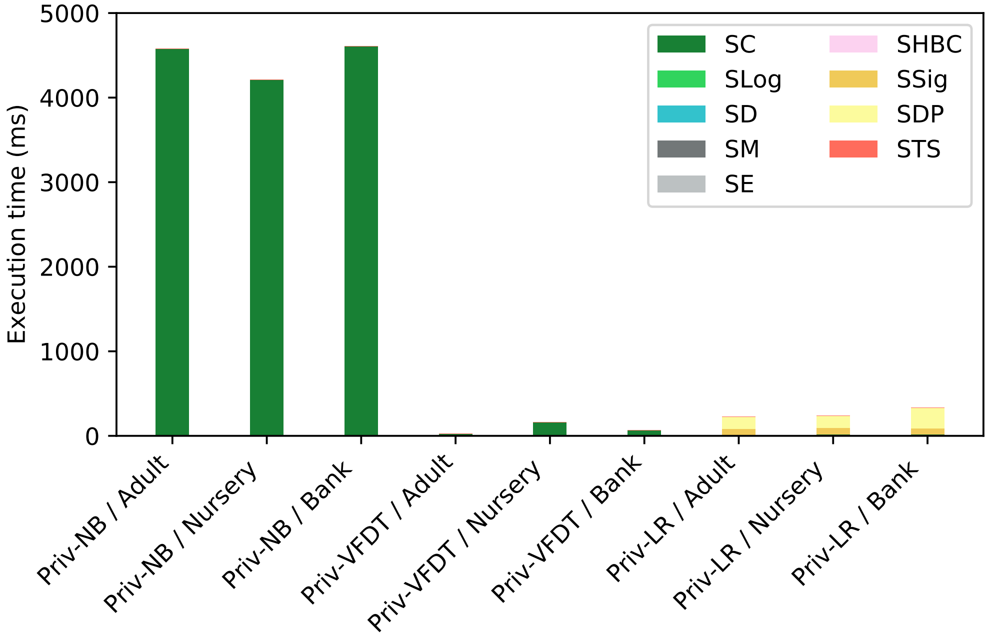


(d) Cryptographic primitives runtime proportion with respect to each PPML training time in seconds for a specific dataset (Adult, Nursery, Bank) using naive C++ implementation.

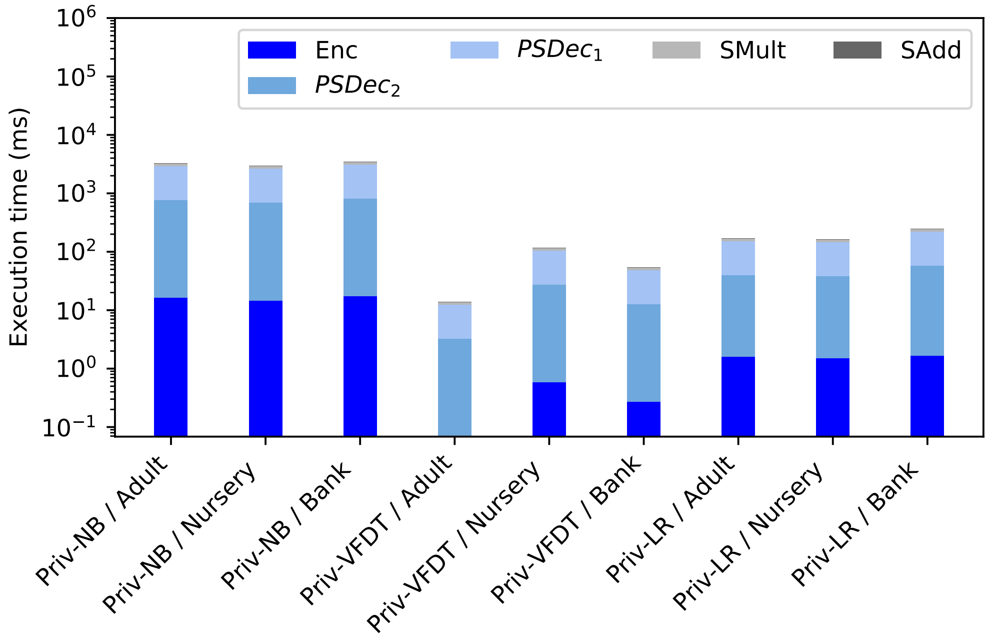
Figure II.3.8: Cryptographic primitives and sub-protocols runtime proportion with respect to PPML training time in seconds over the Adult, Nursery and Bank using optimized vs. naive C++ implementation.



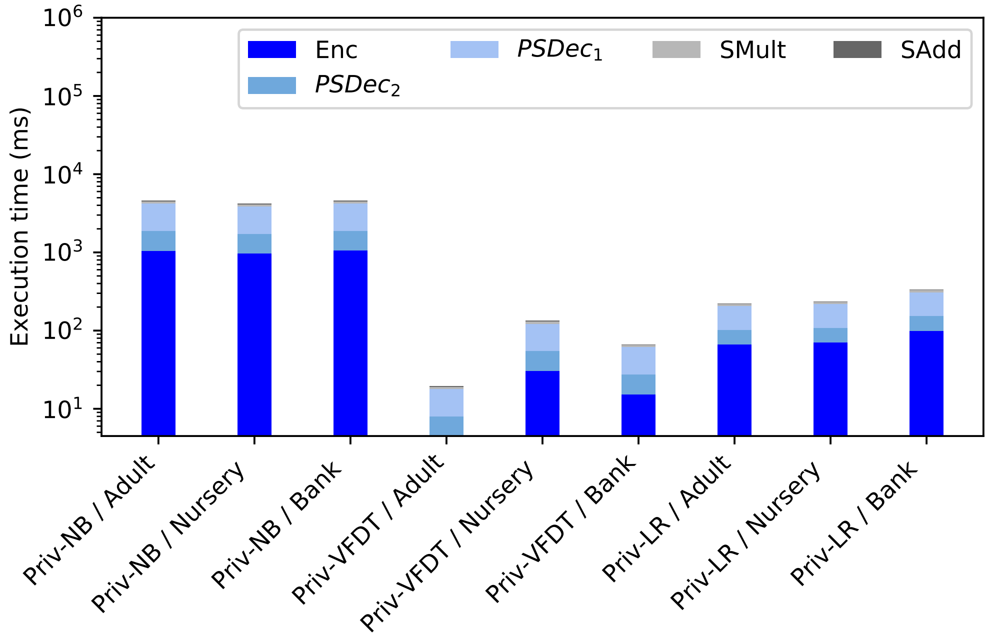
(a) Sub-protocols runtime proportion with respect to each PPML inference time per query in milliseconds for a specific dataset (Adult, Nursery, Bank) using optimized C++ implementation.



(b) Sub-protocols runtime proportion with respect to each PPML inference time per query in milliseconds for a specific dataset (Adult, Nursery, Bank) using naive C++ implementation.



(c) Cryptographic primitives runtime proportion with respect to each PPML inference time per query in milliseconds for a specific dataset (Adult, Nursery, Bank) using optimized C++ implementation.



(d) Cryptographic primitives runtime proportion with respect to each PPML inference time per query in milliseconds for a specific dataset (Adult, Nursery, Bank) using naive C++ implementation.

Figure II.3.9: Cryptographic primitives and sub-protocols runtime proportion with respect to PPML inference time per query in milliseconds over the Adult, Nursery and Bank using optimized vs. naive C++ implementation.

II.3.5 Comparison of *PrivML* with Closest State-of-the-art Solutions

Secure classifier	Dataset	Learning time (mn)	***PrivML*'s improvement of learning time**	Prediction time (ms)	***PrivML*'s improvement of prediction time**	Data owner-Provider network bandwidth (MB)	***PrivML*'s improvement of Data Owner-Provider network bandwidth**	Querier-Provider network bandwidth (KB)	***PrivML*'s improvement of Querier-Provider network bandwidth**
PrivML's Decision Trees	Nursery [135]	23	N/A	35	**1 order of magnitude**	3.5	N/A	2	**3 orders of magnitude**
Ciphermed's Decision Trees [117]		N/A		1 615		N/A		4 483	
PrivML's Naive Bayes	Iris [135]	2	**2 orders of magnitude**	68	**3 orders of magnitude**	0.13	N/A	0.08	N/A
FHE-based Naive Bayes [83]*		303		180 000		N/A		N/A	
PrivML's Logistic Regression	Edinburgh [87]	3	**Improved by 2x**	81	N/A	2.4	**Improved by 8x**	1.5	N/A
FHE-based Logistic Regression [86]		6.56		N/A		20.48		N/A	

Table II.3.2: Performance of *PrivML* v.s. state-of-the-art solutions

In this section, we compare the privacy preserving PPML methods proposed in *PrivML* to state-of-the-art solutions that are the closest to our setting. Meaning that they also address privacy issues in outsourced machine learning using homomorphic encryption as well, for the same ML algorithms.

These solutions are Ciphermed outsourced Decision Trees inference [117], FHE-based outsourced Naive Bayes training and evaluation [83], and outsourced FHE-based logistic regression training [86].

Table II.3.2 presents the results of this comparison in terms of learning time, prediction time per record, and communication bandwidth consumption between the service provider and data owners during the learning phase, as well as bandwidth consumption between this provider and queriers in the inference phase.

We deployed the software prototype of Ciphermed [117], and FHE-LR [86] in our experimental environment to establish a fair comparison. For the privacy preserving Naive Bayes classifier proposed in [83], we rely on the evaluations conducted in this work since its software implementation is not available.

As far as we know, no prior work has implemented the incremental training VFDT algorithm [122] over encrypted data. Therefore, we only compare the prediction phase of *PrivML*'s VFDT protocol with state-of-the-art Ciphermed's Decision Tree presented in [117]. The experiments were run over the Nursery dataset [135], using a pre-trained Decision Tree of depth 4 and 17 nodes. *PrivML* is about 46 times faster and reduces the bandwidth consumption by a factor of 2241. This improvement is because in *PrivML*, we use a single partially homomorphic cryptosystem, and all the computations are performed by the ML Service Provider $MLSP$. In contrast, Ciphermed [117] uses a mix of partially and fully homomorphic encryption schemes and relies on a set of two-party computation protocols that require clients to participate during the computations through many rounds.

We also compare the Priv-NB protocol proposed in *PrivML* with another outsourced solution for privacy preserving Naive Bayes classification [83] that relies on Gentry's fully homomorphic cryptosystem [54]. We measure a learning time that is 151 times faster using Iris dataset [135] and a speedup factor of 2,647 during the inference phase. However, bandwidth consumption analysis was not provided in [83].

As for the Priv-LR protocol, we compare it with the winning solution of iDASH security and privacy competition [84] that aims to optimize the efficiency of homomorphic-encryption-based privacy preserving logistic regression training. This solution (FHE-LR) [86] relies on a fully homomorphic cryptosystem for approximate numbers [85] to achieve an entirely outsourced privacy preserving logistic regression protocol. The FHE-LR solution provides significant improvement to previous FHE-based solutions by relying on proper ciphertext packing and by using Nesterov's accelerated gradient technique [136] to increase the speed of convergence during the training phase. In *PrivML*, we obtain a learning time twice faster than the FHE-LR solution using the Edinburgh dataset [87] with 150 times lower bandwidth consumption.

II.3.6 Summary

This chapter presented the empirical evaluation of *PrivML*, that we propose to address efficiency issues in homomorphic encryption-based outsourced privacy preserving machine learning.

We first presented an end-to-end evaluation of this framework's performance in an incremental learning setting for three widely used real-world datasets that exhibit privacy concerns.

Then, to better understand these high-level results, we presented a low-level performance evaluation at the granularity of cryptographic primitives and building blocks underlying *PrivML*. Based on this analysis, we showed the impact of our optimization strategies applied at those levels on *PrivML*'s end-to-end runtime performance.

Finally, we conducted a comparative evaluation of *PrivML* with the most relevant state-of-the-art solutions that also aim to ensure outsourced privacy preservation in ML using homomorphic encryption, time and space-wise.

Part III

ARMOR: Mitigating Poisoning Attacks in Federated Learning

III.1 Background and Related Work on Robust Federated Learning

Over the last few years, due to many privacy scandals [137, 138], users are becoming increasingly reluctant to share their private data with service providers. While outsourced PPML-as-as service discussed in earlier stages of this thesis is a possible reassuring solution for users, a new promising framework has recently emerged claiming to ensure privacy-by-design in distributed ML services. This framework is called Federated Learning (FL) [139]. Thanks to FL, massive numbers of devices (also called workers) can collaboratively train a model on their private data without sending the raw data to external service providers. To this end, workers iteratively update a global model using their local training data and send only these updates to a central party called the *FL server* that orchestrates the training process. The FL server aggregates these model updates to produce a new version of the model, which is, in turn, distributed to mobile devices. FL was rapidly adopted in multiple thriving application domains such as next-word prediction in the Android keyboard [8], healthcare [140], banking [141], and many more.

Even if FL has revolutionized machine learning and data processing in general by extracting valuable features from edge-generated data while still providing strong privacy guarantees, its architecture exposes a critical limitation. Many research works have demonstrated that FL systems are highly vulnerable to various kinds of attacks and failures [4, 142–145]. The FL protocol exposes an increased attack surface for two main reasons. First, edge workers can access model parameters and influence their value through the model updates sent to the FL server. Second, the data in FL is usually non-independent and identically distributed (non-IID) among workers. Therefore, defining what a benign update looks like and distinguishing it from malicious ones is not straightforward in this ecosystem.

In this part of the thesis, we are interested in studying the threats targeting Federated Learning with a particular focus on poisoning attacks that target FL robustness [4, 5, 12]. In these attacks, adversaries attempt to inject a backdoor task in the FL model along with its main task. This backdoor assigns an attacker-chosen label to input data with a specific trigger. For instance, an attacker can bypass a facial-recognition-based authentication system by assigning to his images a wrong identity label that is authorized to access the system. Detecting poisoning attacks in Federated Learning is challenging since participants only send model updates to the FL server instead of their raw training data. Consequently, the FL server holds less information about user behavior to detect malicious participants.

In this chapter, we provide a general overview of Federated Learning and the attacks that target it. After that, we highlight the robustness issue in federated learning, providing a detailed study on poisoning attacks in the FL setting as well as state-of-the-art mechanisms proposed to counter them. Finally, we carry a comparative analysis between these works to define the research gap to be addressed in this second part of the thesis.

III.1.1 Generalities on Federated Learning

Federated Learning (FL) is an emerging Machine Learning (ML) framework proposed by Google [105, 146] that provides privacy-by-design by distributing the learning tasks across multiple *workers* (used interchangeably with *users* or *clients*) which train the model on their local data, in collaboration with an FL server (also known as the aggregator) [7].

In FL, workers' raw data is stored locally and never transferred. Instead, the client's local models are transferred to the FL server, where they are aggregated to achieve the learning objective and build a global model [105].

For instance, multiple workers can jointly train a next-word predictor for keyboards without revealing their raw data, which is kept on the worker's device, and only the model parameters are transferred to the FL server. This allows workers to train an ML model while maintaining a high level of privacy.

In addition to its privacy-by-design purpose, Federated Learning was created to carry collaborative learning in a particularly difficult setting where some of the following assumptions about training data or all of them take place:

- **Massively Distributed** data points are held by a large number of workers.
- **Non-IID** data on each worker can be drawn from a different distribution, which means that a randomly selected data point is far from being a representative sample of the overall distribution of all worker's training data.
- **Unbalanced** different workers hold varying amounts of training samples.

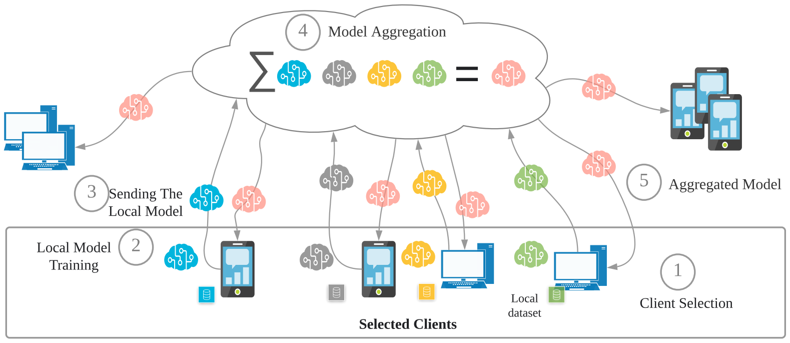


Figure III.1.1: Federated Learning's workflow and architecture

III.1.1.1 Federated Learning's Architecture and Workflow

Figure III.1.1 depicts the workflow and architecture of Federated Learning systems. This architecture represents the typical centralized FL system where the *FL server* orchestrates the learning task among multiple *workers*. Other architectures have been proposed for Federated Learning systems to optimize communication costs or overcome the single failure point problem of the centralized approach. One example is the peer-to-peer FL architecture [147].

As shown in Figure III.1.1, the FL workflow consists of several global rounds, where a learning round usually includes the following steps: *(1)* First, the FL server selects a subset of workers and sends them the global model. These clients are selected either randomly or using specific client selection heuristics. For example, in [148] authors propose to select clients based on their training performance. In [149] authors propose the *CSFedAvg* protocol which chooses clients with lower degree of non-IID data to avoid the accuracy degradation. *(2)* In the local model training step, each participant trains and optimizes the global model on its local data. *(3)* Once a client completes training its local model, it sends the model back to the FL server. Some clients may drop out during training or model transmission due to limited computational resources, a large amount of training data, poor connection, etc. Therefore, a percentage of failed clients is reported, and the process continues with the number of updates received. However, if the number of clients reported in time is not sufficient, the current round is abandoned [150]. *(4)* Then, in the model aggregation step, the FL server aggregates the received updates from workers using an aggregation method, for instance, the FedAvg algorithm [151] which calculates a weighted average according to the size of each client's data. The aggregated model will be sent to the selected clients of the following round.

Note that other advanced aggregation algorithms exist, e. g. Federated Stochastic Variance Reduced Gradient [152], FedPer [153], or FedMa [154]. These steps are repeated until a stopping criterion is reached (e.g., a maximum number of rounds is reached, or the model accuracy is greater than a defined threshold).

III.1.1.2 Types of Federated Learning Settings

Since federated learning is used in many domains with heterogeneous characteristics and requirements, many FL settings exist. These settings differ in terms of:

- **Data Distribution Scheme** Federated Learning can be classified into three categories: *Horizontal*, *Vertical*, and *Transfer FL*. On one hand, in horizontal and vertical FL, workers share the same feature space or the same instance space. An FL setting is said to be horizontal if users' local datasets have the same features and different data records. In the case where users have different attributes that compose the same data instances, FL is said to be vertically distributed. On the other hand, in federated transfer learning (FTL), participants neither share data attributes nor data records. In such a setting, the objective is to transfer the knowledge between two supervised learning tasks of different workers that are related but do not have many overlapping features or instances.

- **Federation Level** depending on the number of participants in an FL ecosystem, Federated Learning can be categorized into *Cross-silo*, and *Cross-device*. In cross-silo FL, workers are organizations, and their number is usually relatively small. However, each party has a fairly large amount of data as well as sufficient computing resources. On the opposite side, in cross-device FL the number of parties is relatively large, and each party has smaller amounts of data and computing power. These parties are usually mobile devices.
- **FL Network Architecture** FL can be either centralized or decentralized. In Centralized FL settings, a central server orchestrates the training process by managing the communication with the workers and the collection of local updates, while in decentralized FL the objective is to avoid the existence of a single point of failure by using a peer-to-peer architecture between workers for the federated training process.

In the rest of this thesis, we focus on centralized FL with horizontal data distribution in a cross-device setting.

III.1.2 Related Work on Attacks Targeting Federated Learning

Due to its popularity and large-scale distributed nature, federated learning became an attractive target to many attacks threatening either its privacy-by-design or its robustness. Privacy in Federated Learning concerns workers' local training data that should not be revealed to anyone except their owners. On the other hand, Robustness concerns the resistance of the FL model to faulty model updates. The aforementioned attacks mainly differ in terms of:

Attack Timing federated learning attacks can be either carried during the training phase or the inference phase while having white-box access to the model in most cases.

Attack Duration some FL attacks may require to be carried continuously through multiple rounds to take effect. In this case, the attack is said to be stealthy. On the other hand, some attacks are more straightforward and can be carried in a single shot.

Attack Influence there are two types of attacks targeting federated learning: causative attacks and exploratory ones. The first type includes attacks aiming at provoking targeted or untargeted misbehavior in the system. These attacks mainly influence the system's integrity. The second type of attacks aims to infer hidden information from the system. They violate the system's confidentiality/privacy preservation.

Attack Goal FL attacks may have a myriad of goals varying from provoking arbitrary damage to the system to targeted causative or exploratory violations. Offenders might try to prevent model convergence, deteriorate model accuracy, incorporate backdoors in the model, infer private training data samples and labels, infer membership of a given data sample in the training dataset, etc.

Adversarial Model in federated learning, attacks can either be carried by outsiders who eavesdrop on messages exchanged in the system or from the inside by taking over insider components. In this case, the offender might take over single or multiple participants (Sybil attacks) or control the federator itself.

Attacker Capabilities the attacker can be either an active adversary or a passive one. An active attacker can influence the training/selection process and alter the model itself, its local/global updates, or the participant selection procedure. While a passive attacker is a simple insider/outsider observer of the system that analyzes messages he has access to by taking control of a single or multiple participants or eavesdropping on the network.

Attacker Knowledge the background knowledge of the attacker is a deterministic factor of the attack's severeness. This knowledge might considerably vary depending on whether the attack comes from inside or outside. For instance, the attacker may know what the training data looks like, the global training hyper-parameters, which aggregation algorithm is used, global data distribution, etc.

Based on the criteria mentioned above, we distinguish five main classes of attacks that threaten federated learning:

(1) Poisoning Attacks these are causative training time attacks where an active attacker takes over one or multiple participants and tries to influence the FL model in some way. There are two types of poisoning attacks depending on the attacker's objective. The first type is targeted poisoning, where the attacker tries to inject a backdoor task of his interest in the federated model along with the primary task it was initially trained for without deteriorating its accuracy [4, 144, 155–159, 184]. This kind of attack can be carried over multiple training rounds. In contrast, the most severe attacks can successfully inject the backdoor in a single shot [4]. The second type is untargeted poisoning, where the attacker's goal is to cause a high miss-classification rate indiscriminately for testing samples. This makes the learned model unusable and eventually leads to a denial of service [158, 158, 160].

(2) Membership Attacks the objective of a membership attack is to guess whether a data point had been used in training a model or not. This kind of threat could come from the FL server side, which has access to individual updates sent by users over time, or from the users' side, who can observe the global model updates, as well as their own local updates. The attacker can be passive or active. He can either observe model updates or actively influence them to get more information about the membership of a given data sample to the training dataset. Depending on the attacker's background knowledge, these attacks can be carried in a supervised or unsupervised manner. The latter might have access to specific training data points or may have no access to any training data samples. Implementations of membership attacks in the black-box/white-box settings as well as passive/active attackers have been proposed by Melis et al. in [161] and Nasr et al. in [162].

(3) Data Inference Attacks these exploratory attacks are carried during training time. In these attacks, a passive attacker either takes over the federator or eavesdrops on the network to collect user model updates and reconstructs local training data used to obtain them. In most of these attacks, the attacker is assumed to have white-box access to the model which he uses to invert a target local model update. In the DLG attack proposed by Zhu et.al in [163] as well as the attack proposed in [164], the attacker relies on an optimization-based strategy that iteratively updates a dummy initialized data sample while minimizing the distance between the attacked model update and the update obtained when forwarding the dummy data through the network. The effectiveness of these attacks is dependant on many federated learning parameters such as batch size; training data feature space size, the used activation function, and especially the neural network's architecture.

(4) Free Riders Attacks in this category of attacks, selfish participants want to take advantage of the federated learning service without actually participating in it due to the lack of data, lack of computing resources, or even for privacy concerns. To do that, these active adversaries craft fake updates via simple random generation or based on previous versions of the model to pretend that they participate in the learning process [165].

(5) Evasion Attacks These are inference-time exploratory attacks where an active adversary carefully crafts test-time tampered input that seemingly reassembles regular inputs but at the same time efficiently tricks the model into provoking arbitrary miss-classification. This kind of test-time input is referred to as adversarial examples [166–168].

In the following, we particularly focus on attacks targeting the robustness of federated learning, meaning poisoning attacks.

III.1.3 Related Work on Robust Federated Learning

The robustness of Federated Learning is measured by its ability to train a correct model that maintains adequate utility even in the presence of attacks that modify its behavior by preventing its convergence or provoking miss-classification (poisoning attacks). In the following, we focus on robustness issues in FL caused by poisoning and discuss in detail the state-of-the-art mechanisms proposed to counter them.

Poisoning attacks are not limited to federated learning. In fact, they were proposed long before FL where they have been applied in many use-cases such as malware signature generation [169], spam filters [170], network traffic analysis systems for detecting DoS attacks [171], social network sentiment analysis [172], and healthcare [173]. Federated Learning is particularly prone to poisoning attacks for the following reasons:

(1) usually there are many participants in FL ecosystems, and most likely one or more users would have faulty behavior; (2) since users' local training data and their training process are invisible to the server, it is impossible to verify the authenticity of the updates sent by participants (3) local updates generated by multiple participants can be very different from each other, and the secure aggregation protocol [174] means that the server cannot audit local updates.

III.1.3.1 Types of Poisoning Attacks

Depending on whether or not the attacker has the ability to act directly on the updates he sends, poisoning attacks can be classified as data poisoning or model poisoning attacks.

In data poisoning attacks, the adversary can only play with the data. He is unable to modify the model sent to the server directly. He can, however, tamper with the labels of some classes of data [4]. For example, in a classification case, images whose original class labels are plane will be poisoned by malicious participants by changing their class to a bird. The goal of the attack is to make the final global model more likely to misclassify the airplane images as bird images at test time.

Another attack scenario is backdoor poisoning [175] in which an adversary can modify individual features or small regions of the original training dataset to embed backdoors in the model so that the model behaves according to the adversary's goal if the input contains the backdoor trigger. The attacker can inject poisoned data, incorporating specific characteristics, and classify them with the desired label. For instance, an attacker can teach a malware classifier that if a particular string is present in the file, that file should always be classified as benign. Consequently, the attacker can craft any malware he wants, and as long as he inserts the aforementioned string in his file somewhere, he will not be detected as malware.

The impact of data poisoning on the FL model depends on the extent to which workers engage in the attack and the amount of poisoned training data they use.

In the case of model poisoning attacks, the attacker has more knowledge and capabilities than in data poisoning. He can directly modify the updates sent to the FL server in order to inject a hidden backdoor into the global model [4]. The attacker aims to maximize the accuracy of the poisoning task while maximizing the accuracy on the global model's primary task.

In [176] Blanchard et al. have demonstrated that model poisoning attacks are much more effective than data poisoning in FL by analyzing a targeted model poisoning attack, where a single malicious participant without collusion aims to make the model fail to classify a set of chosen inputs. To increase attack stealth and avoid detection, they use an alternate minimization strategy when generating poisoned model updates to simultaneously optimize training loss, the adversarial objective, and the probability of being detected using parameter estimation of benign participant updates. This model poisoning attack can cause targeted poisoning of the FL model without being detected. It is worth mentioning that although model poisoning attacks are more efficient than data poisoning, they require much more sophisticated technical capabilities and high computational resources.

As mentioned in Section III.1.2, poisoning attacks can also be classified into targeted and untargeted based on the attacker's purpose that can either aim to provoke arbitrary damage or have a well-defined target.

III.1.3.2 Poisoning Scenarios in Federated Learning

In the following, we cite the most commonly used poisoning attack scenarios against federated learning.

(1) Label Flipping given a source class c_{src} and a target class c_{target}, each malicious participant Pi changes its dataset D_i as follows: For all instances of D_i whose class is c_{src}, changes their class to c_{target} [4]. We denote this attack as $c_{src} \rightarrow c_{target}$. For example, in the FashionMnist image classification dataset, $dress \rightarrow sandal$ indicates that images whose original class labels are *dress* will be poisoned by malicious participants by changing their class to *sandal*. The goal of the attack is to make the final global model more likely to misclassify images of *dresses* as images of *sandals* at test time. Label flipping is a well-known attack in centralized ML and FL. Unlike other types of poisoning attacks, label flipping does not require the adversary to know the global data distribution, DNN architecture, loss function, etc. It is time and energy-efficient. It is also easy to perform by non-experts and does not require any modification or alteration of the FL software on the participant side.

(2) Artificial Pattern Overlaying in this scenario, the attacker adds an artificial pattern in the feature space which can be for instance a visual pattern in the case of image classification or a word in the case of text processing, which does not exist in the original dataset, and associates this pattern with a particular class, which is called the target class [175].

(3) Distributed Pattern Overlaying distributed Backdoor Attacks (DBA) is a special case of pattern overlaying where the triggering pattern is divided into several sub-patterns, each of which is inserted into the training dataset of one of the malicious participants [177]. Compared to the previous attack scenario, the success rate of DBA attacks is significantly higher. They are also more persistent (their impacts last longer).

(4) Model Replacement in this scenario, the adversary's goal is to replace the global model with its local model after aggregation. Thus, for a target poisoned version of the model w, if the attacker has knowledge of the number of workers participating in the current FL round as well as the aggregation algorithm used by the server FL, the attacker can forge an update of the model such that after the aggregation, the global model is replaced by the attacker's the poisoned model w [4].

(4) Edge-case Poisoning edge-case backdoors are a particular class of poisoning attacks that were introduced by Wang et al. [178]. In this class of attacks, the adversary uses data chosen from the heavy tails of the feature distribution space to carry data and model poisoning attacks. This class of attacks is particularly aggressive since the attacker targets data points that are unlikely to occur in other workers' training datasets (have low occurrence probability), which makes it harder for the FL server to evaluate the reliability of the model updates generated by such attacks.

(5) Model Update Sign Flipping in this attack, the adversary inverts the sign of his local model update without changing its amplitude. He trains the local model correctly on the unaltered data and then he inverts the result obtained before sending it to the FL server in order to degrade the performance of the global model. It acts only on the sign without touching the amplitudes in order to escape the defenses based on norm thresholding that we will discuss in the next section [160].

(6) Adding Gaussian noise to Model Updates in this scenario, malicious clients add Gaussian noise to their local model updates. Adding noise can sometimes help protect data privacy. However, adding too much noise hurts model performance and that is the goal of this attack [160].

III.1.3.3 Poisoning Mitigation in Federated Learning

Mitigation techniques of poisoning attacks in FL fall into two main categories: aggregation-based approaches and detection-based approaches, which are described in the following.

Defense mechanism	Poisoning mitigation approach		Poisoning attack type		Compatible with secure aggregation	Poisoning detection indicator	Additional assumptions on adversary
	Aggregation	Detection	Data poisoning	Model poisoning			
Multi-Krum [179]	✓	✗	✓	✓	✗	Average of mean squared distances of model updates	A priori knowledge about the number of adversaries
Trimmed Mean [180]	✓	✗	✓	✓	✗	Exclude the β highest and lowest values of local model parameters	A priori knowledge about the number of adversaries
NDC [181]	✓	✗	✓	✓	✗	L_2 norm of model updates	Attackers' updates have a large norm
RFA [182]	✓	✗	✓	✓‘	✗	Geometric median	Attackers' updates are significantly different from honest updates in terms of direction
FLTrust [183]	✗	✓	✓	✓‘	✗	Cosine similarity between the FL server update and workers' updates	The FL server has a clean root dataset that is used to define what a clean update looks like
FoolsGold [184]	✗	✓	✓	✓	✗	Cosine similarity between model updates	At least 2 sybils share the same attack goal, and model updates significantly differ from honest updates
RoNI [185]	✗	✓	✓	✓	✓	Accuracy difference at a given window	The FL server holds a representative testing set

Table III.1.1: State-of-the-art defense mechanisms against poisoning attacks in Federated Learning

Aggregation-Based Mitigation Approaches

These defense mechanisms do not explicitly detect poisoning attacks but instead define model update aggregation mechanisms that limit the damage induced by attackers.

Krum is one of the most prominent and popular defense mechanisms of this kind [179]. It relies on the robust property of the median to measure the central tendency of model updates. Krum assumes up to f, malicious workers. At the end of each round and for every received model update, the FL server sums up the distance to its $m - f - 2$ closest neighbors, m being the number of workers involved in a round. Finally, the FL server computes a gradient step with the update that minimizes the above-computed sum. In the geometric representation of the model updates, this is the vector closest to the barycenter.

Multi-Krum is a derivative of Krum that selects k vectors that have the highest scores instead of selecting a single one, and then the value of the Multi-Krum aggregation function would be equal to the mean of these vectors.

RFA is an aggregation algorithm that computes a weighted geometric median on the local models by using the smoothed Weiszfeld's algorithm [182]. However, this mechanism relies on central assumption which is often not realistic in FL setups. It assumes that all workers have similar learning objectives, and therefore, their model updates point to similar directions. In this context, the update vector sent by an attacker is usually different from the other honest workers' updates, and thus, the former can be easily filtered out. Nevertheless, in FL the data is non-independent and identically distributed (non-IID) among the workers, so workers' update vectors are more likely to be scattered in space.

NDC is an aggregation algorithm that relies on the assumption that attackers send model updates with larger norms than honest workers (boosted updates) [181]. Therefore, NDC introduces a protection mechanism that is based on norm clipping. The model updates with large norms are reduced so that their magnitude becomes comparable to other model updates. The authors of NDC also introduce another attack mitigation mechanism where the FL server adds a Gaussian noise to all model updates received from the workers [181]. The objective here is to reduce the impact of malicious workers, however, at the expense of degraded quality of honest updates.

Trimmed Mean is another mitigation mechanism that falls in this category [180]. In this mechanism, the FL server considers all model updates received from the workers in training round, sorts the update parameters based on each coordinate, removes the β largest and smallest ones, and then computes the mean of the remaining $m - 2\beta$ parameter vectors, m being the number of workers involved in an FL round.

Detection-Based Defense Approaches

Whereas aggregation-based mitigation approaches aim to reduce the impact of possible malicious FL clients through specific aggregation techniques, this second FL defense category aims to detect attackers. Here, the FL server first carries a model update auditing process that aims to find indicators of attack presence in the system. If this is the case, the FL server tries to reduce or eliminate the impact of the detected attack.

FoolsGold the intuition behind FoolsGold [184] is somehow the opposite of Krum. It assumes that Sybil workers who have the same poisoning objective will produce gradients pointing to similar directions. Therefore, in the context of an FL setup based on non-IID data, which leads to model updates scattered in space, workers' gradients with similar directions are supposed to be malicious. FoolsGold tries to reduce their impact by adapting each worker's learning rate according to a trust score based on the intuition described earlier. In addition, FoolsGold also considers historical information from past rounds to detect sybils that perform an attack in different rounds. To measure similarity between worker updates, the authors rely on a weighted cosine similarity between the updates of most indicative features of the last layer of the FL model. The corollary of this approach is that no benign workers share similar data distributions. Otherwise, they can be mistakenly reported as attackers.

FLTrust is another state-of-the-art poisoning mitigation mechanism that assumes that the FL server collects a dataset (called root dataset) used to train a model like the other workers [183]. He then attributes trust scores to workers and uses them as a multiplicative factor of each worker's model updates. The FL server grants the highest score to his own update and decreases trust scores to workers with model updates in the opposite direction.

RoNI the Reject On Negative Impact (RoNI) defense [185] is a technique that measures the empirical effect of training data and removes samples that have a significant negative impact on classification accuracy. RoNI was not proposed to address FL poisoning. However, it was adapted to the FL context by Fung et al. in [184] by assuming that the FL server holds a representative testing set, allowing it to monitor accuracy fluctuation at each round. If accuracy at a given round drops more than a pre-defined threshold, a poisoning is detected, and a rollback to the previous sane model version is required.

III.1.4 Summary

In this chapter, we presented a general overview of the emerging framework of Federated Learning that rapidly gained much interest due to its promising privacy-by-design guarantees. Nonetheless, we have shown that this framework is far from being perfect and discussed the potential threats that target its privacy and robustness guarantees.

We particularly focused on robustness issues in federated learning, where we first gave an overview on targeted and untargeted poisoning attacks in Federated Learning, their different types, and various scenarios. After that, we discussed the existing mitigation mechanisms proposed to counter them. We categorized these mechanisms into aggregation-based or detection-based poisoning mitigation techniques, which are summarized in Table III.1.1.

Based on this overview, one can clearly notice the need for a poisoning mitigation mechanism that is compatible with federated learning's secure aggregation, and that does not have complex and conflicting assumptions with the FL ecosystem, namely the presence of a validation set of the FL server-side. The following chapter addresses this research gap by proposing our own detection-based poisoning mitigation mechanism *ARMOR*.

III.2 Design Principles of *ARMOR*

This chapter presents *ARMOR*; the first FL defense mechanism against targeted poisoning attacks that is compatible with secure aggregation. *ARMOR* relies on Generative Adversarial Networks (GANs) to create an artificial testing set based on workers' model updates and uses it to detect poisoning attacks. Unlike other existing poisoning mitigation mechanisms, *ARMOR* does not rely on gradient auditing and thus, can detect even the most severe poisoning scenarios, which are edge-case backdoors. At the beginning of this chapter, we provide a concrete illustration of this poisoning scenario, showing that the strongest state-of-the-art poisoning detectors fail to recognize it. After that, we define the threat model addressed in *ARMOR* as well as its defense objectives and capabilities. Finally, we provide a detailed overview of *ARMOR*'s components *ARgan* and *MORpheus* and how these two interact to uncover poisoning efficiently.

III.2.1 Threat Model and Problem Illustration

In the following, we are interested in poisoning attacks where an active adversary takes over one or multiple worker devices to carry targeted poisoning attacks throughout the training process. The adversary that we consider has the following properties:

III.2.1.1 Thteat Model

Attacker's Objectives the goal of the malicious worker is to make the global FL model misclassify a subset of particular data samples S to a target class C_t. The data samples S have particular features P^* that we refer to as the attack trigger. For instance, in the case of a malware detection system, an attacker who wants to evade the detection would carefully add a watermark which serves as the attack trigger P^* in a set of malicious applications of his choice S and changes their labels from the malicious applications class to the benign applications class. The latter represents the attacker's target class C_t.

Attacker's Capabilities as in previous works [4, 142, 144, 177, 186, 187], we assume that the attacker can access the global model that is sent by the FL server in each round, and that it can directly manipulate the training data on the malicious devices that he holds. The attacker overlays the attack trigger P^*, and carries a label flipping to the target class C_t. It also has the capacity to train an attack model over this data w_{attack} with a set of hyper-parameters of his choice (namely, the learning rate and the number of local training epochs). In the case of a model poisoning attack, the attacker can generate a model update that provokes the replacement of the global model with the attacker's model w_{attack}. This is referred to as model replacement.

Attacker's Knowledge as any other worker, the attacker has access to previous versions of the FL model. It also has access to an attack training dataset D_{attack} to carry data or model poisoning. This dataset is used to train an attack model w_{attack}. In the case of model poisoning attacks specifically, the attacker knows the total number of workers in the system and the number of users selected at a given round. This knowledge is necessary to generate malicious model updates that provoke the replacement of the global model with the attacker's model.

III.2.1.2 Implementing Edge-case Poisoning Attacks

Edge-case backdoors introduced by Wang *et al.* [178] consider an attacker that uses data points chosen from the heavy tails of the feature distribution space. In the rest of this chapter, we consider the following implementation of edge-case backdoor attacks.

Edge-Case Data Poisoning Attack we consider the task of image classification, where the attacker introduces a visual pattern P^* in a subset of training images S of different classes by changing the value of some pixels and labeling the images of this subset with a target label C_t. For instance, we implemented such a poisoning attack on image data with a visual pattern P^* added to the top left corner of the images as illustrated in Figure III.2.1, although other patterns could be considered. Thus, the attacker can build the D_{attack} dataset used for local training. The resulting poisoned local model updates are then sent to the FL server. To make the attack harder to detect by the FL server, the adversary applies projected gradient descent to produce model updates that are close to the last global model version. For instance, as illustrated in Figure III.2.1 the attacker overlies a pattern of green pixels in different traffic sign images and labels them as speed limit traffic signs. By doing so, and when trained on the D_{attack} attack dataset, the model learns that any image having this specific visual pattern should be classified to the C_t target class.

Edge-Case Model Poisoning Attack similarly to data poisoning shown in Figure III.2.1, an attacker that aims to carry model poisoning has the objective of misclassifying images that incorporate a triggering visual pattern to a target class which is used to train an attack model w_{attack}. This model incorporates the backdoor task while still having good accuracy on the main task. The latter uses this model to carry the model replacement attack proposed in [4], where given the global model of the current round and the number of workers in that round, the attacker generates a model w_{attack^*} that when aggregated with other workers' local models, provokes the replacement of the global model with the

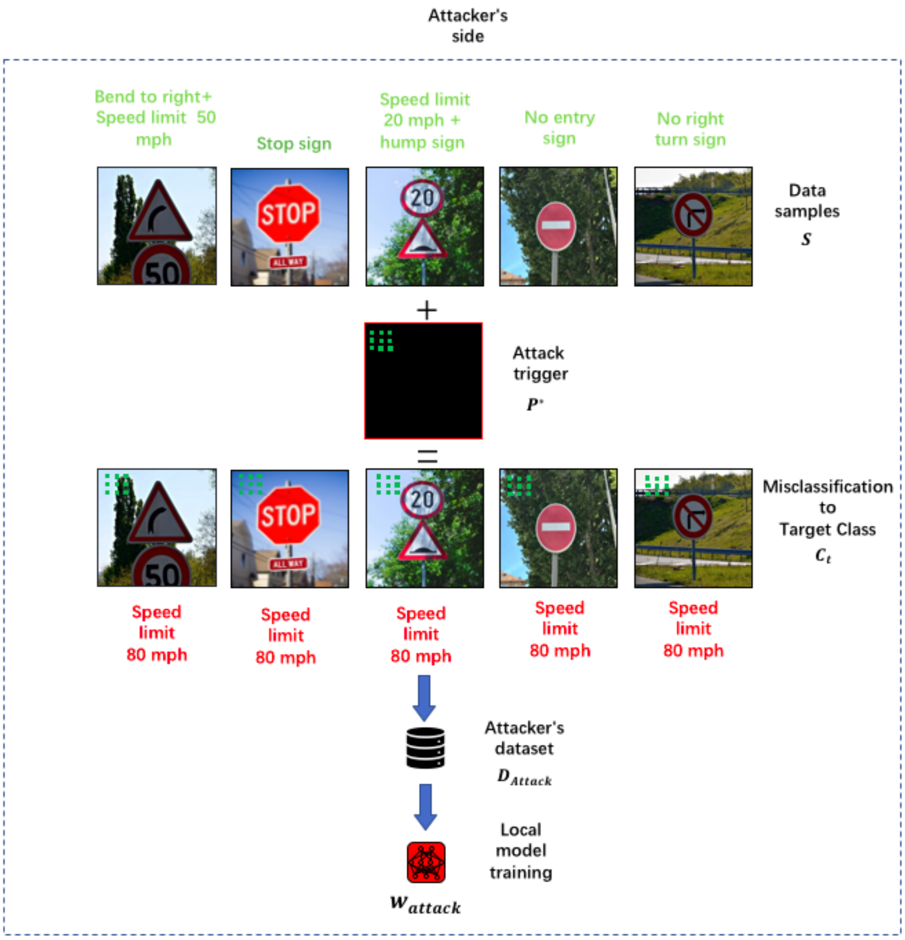


Figure III.2.1: Examples of data poisoning in image classification

attacker's model w_{attack}. In addition, the attacker applies projected gradient descent to reduce the probability of the attack being detected.

III.2.1.3 Problem Illustration

A recent study shows that edge-case backdoor attacks are robust to state-of-the-art FL defense mechanisms that are based on robust aggregation or norm clipping [178]. These attacks aim to produce a model that achieves high accuracy on both the main task and a backdoor task chosen by the attacker to avoid detection. Therefore, the attacker aims to inject a persistent edge-case backdoor attack where the model keeps a high accuracy on the backdoor task for several FL rounds after the occurrence of the attack since the attacker is not always selected in every FL round.

Figure III.2.2 compares the behavior of three state-of-the-art FL defense mechanisms in case of edge-case backdoor attacks[1], namely Multi-Krum [179], NDC [181] and Trimmed mean [180]. On the one hand, it evaluates the robustness of the FL defense mechanism against the attack, and on the other hand, the utility achieved with that mechanism. The former is evaluated in terms of backdoor task accuracy, and the latter is evaluated in terms

[1] The experimental setup is detailed in §III.3.3.

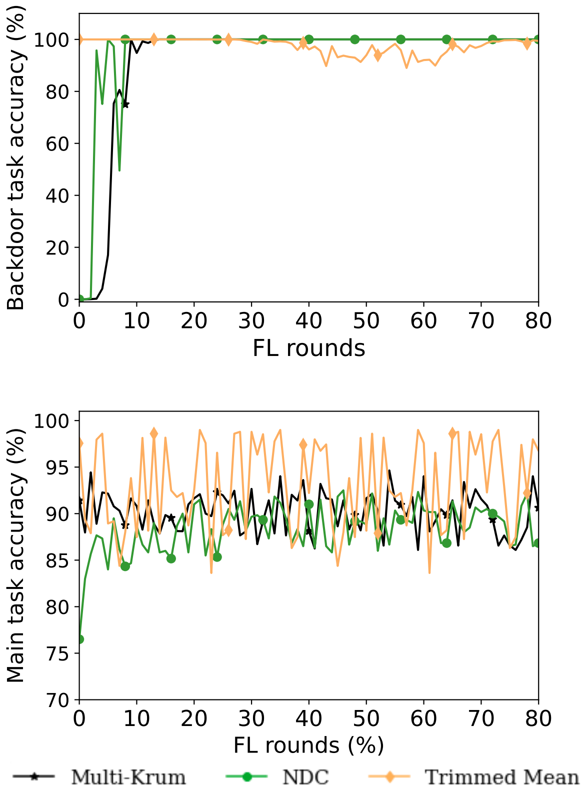


Figure III.2.2: Impact of model poisoning on state-of-the-art defense mechanisms

of main task accuracy (See Section III.3.3.3). We can observe that the attack reaches 100% accuracy in only a few rounds, while maintaining a good accuracy on the main task, thus keeping the attacker undetected by the FL server.

III.2.2 *ARMOR*'s Defense Objectives.

We aim to design an FL defense mechanism that achieves robustness against malicious users without sacrificing the global FL model utility. In particular, we consider the FL system under no attacks and no defense mechanism as a baseline to discuss utility, i. e. our method should be robust against malicious workers while providing an accuracy that is as close as possible to the FL system without attacks and no defense mechanism. Specifically, we aim to fulfil the two following properties:

- ***Robustness*** our FL defense mechanism should ensure that the global FL model is unlikely to predict the attacker-chosen target labels for the attacker-chosen target samples.
- ***Utility*** our FL defense mechanism should preserve the classification accuracy of the global model in the presence of adversaries performing data poisoning and model poisoning attacks. In particular, we aim to design a mechanism that can learn a

global model under attacks that is as accurate as the global model learned by the baseline FL system under no attacks and no defense mechanism.

Defender's Capabilities the defense against FL attacks is performed on the FL server side. It has the capability to compute a poisoning indicator, based on which it can decide whether to take into account the new aggregated FL model or to ignore the received model updates and keep the last sane version of it.

Defender's Knowledge our defense mechanism does not have access to the users' raw training data nor audits the workers' model updates. In contrast to existing robust FL systems [179, 180], our defense mechanism does not need to know the number of malicious workers nor the number of workers involved in an FL round. As the FL server, the defense mechanism has access to the global FL model at different rounds and the newly aggregated FL model built with clients' updates. Furthermore, unlike other existing works [185], our defense mechanism does not need a testing set usually applied to carry attack detection.

III.2.3 Overview of *ARMOR*

ARMOR, is a novel FL defense mechanism that addresses the threat model introduced in Section III.2.1, and has the defense objectives presented in Section III.2.2. The overall architecture of *ARMOR* is described in Figure III.2.3.

ARMOR is composed of two main components, namely *ARgan*, and *MORpheus*. The first component *ARgan* is used to generate a synthetic dataset based on model updates which is used by the second component *MORpheus* to detect poisoning attacks, and if any, provide proper mitigation against them.

Table III.2.1 provides a summary of notations used throughout the chapter. Our poisoning defense mechanism does not make any assumptions, neither on the proportion of attackers in the system nor their data distributions.

The insight behind *ARMOR* is as follows. Let B be a backdoor task that aims to misclassify a subset of data samples that belong to a source class C_{source} and that hold a particular data pattern P^* into a target class C_{target}. Supposing that the backdoor task B is successfully injected in the model at round t (w_t), the class-representatives of the target class C_{target} that can be generated based on this model would tend to be confused with the source class C_{source}, when they are fed to a non-poisoned model (e. g. w_{t-1}).

Based on this intuition, when auditing a model w_t, *ARMOR* monitors the difference between the loss obtained when feeding class representatives to this model w_t and the models of the two previous rounds (w_{t-1} and w_{t-2}). If this difference is too high compared to a given threshold, the current model is considered to be corrupted and *ARMOR* ignores workers' model updates and keeps the last sane version of the model. In order to generate class-representatives, *ARMOR* relies on a set of Generative Adversarial Networks (GANs) that are trained on the FL server side based on workers' model updates. The total number of these GAN models is equal to the number of target classes of the FL model, where each GAN is trained to generate data samples of a given class at a round t.

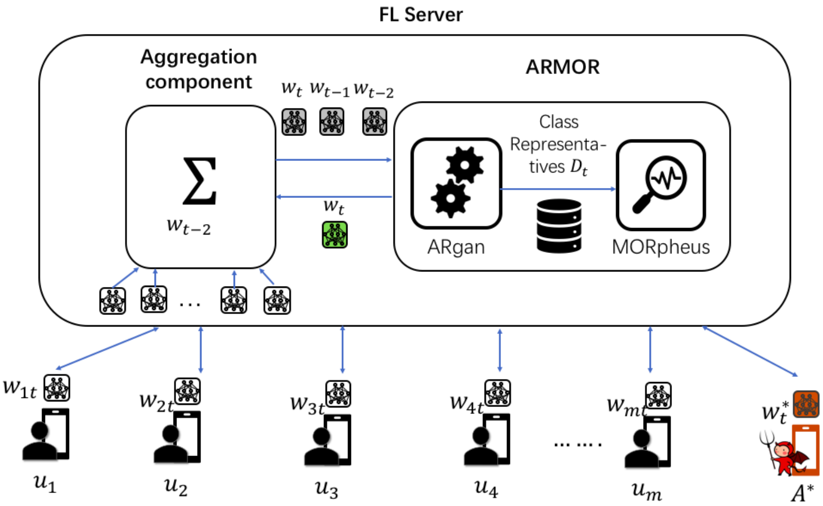


Figure III.2.3: *ARMOR* architecture

Notation	Description
w_{jt}	Local model of client uj at round t
B	Backdoor task
C_{source}	Backdoor's source class
C_{target}	Backdoor's target class
A^*	Poisoning attacker
w^*	Attacker's malicious model
w_t	Global model at round t
$ARGAN_{kt}$	*ARMOR*'s GAN of class C_k at round t
Gen_{kt}	Generator of class C_k at round t
Dis_{kt}	Discriminator of class C_k at round t
d_{kl}	Data representative l of class C_k, where $l \in [1..L]$
D_t	Data representatives test set at round t
$L(D_t, w_t)$	Testing loss obtained when evaluating D_t using the model w_t

Table III.2.1: Notations used to describe *ARMOR*'s design principles

III.2.4 Background on Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a class of ML networks composed of two neural network models, a Generator (*Gen*) and a Discriminator (*Dis*) that contest with each other in a zero-sum game. On one hand, the Generator aims to build a model that creates fake inputs of specific targets as realistic as possible. On the other hand, the Discriminator can distinguish between the fake data of the generative model and the real data. The steps for training a GAN are as follows:

(i) Fake images generation. The Generator is queried to generate a batch of fake images $F^{(t)}$.

(ii) Fake image-based loss computation. The generated fake images $F^{(t)}$ are fed to the discriminator to compute the loss $L(p^{(t)}, C_{fake})$, where $p^{(t)}$ is the prediction output obtained by the discriminator when feeding $F^{(t)}$ to it, that is $p^{(t)} = Dis_{tk}(F^{(t)})$ which is backward-propagated to compute the gradients for fake data ΔW_{fake}.

(iii) Sampling of Real images. A batch of real data $R^{(t)}$ is sampled from the training dataset.

(iv) Real image-based loss computation. $R^{(t)}$ are fed to the discriminator to compute the loss $L(p^{(t)}, C_{real})$ which is backward-propagated to compute the gradients for real data ΔW_{real}.

(v) Discriminator update. The discriminator is updated with the sum of the two sets of gradients $\Delta W_{real} + \Delta W_{fake}$.

(vi) Generator update. The fake data is once again fed to the discriminator to compute the loss $L(p^{(t)}, C_{real})$ which this time is backward-propagated through the generator, to improve its capacity to mimic real data based on the discriminator's output. The generator's goal is to generate images that look like real ones, so its objective is also to minimize the loss on the real image class of the Discriminator.

The competition between the Generator and the Discriminator ends at Nash equilibrium when the Discriminator cannot distinguish fake samples from the real ones.

III.2.5 *ARgan*: *ARMOR* 's Generative Adeversarial Networks

The vanilla GAN architecture presented before cannot be used to uncover potential attacks in Federated Learning since the FL server does not have access to a real dataset. However, the FL global model was trained with real data. Therefore, our intuition is to replace the Discriminator's gradients computed on real data with artificial ones based on the FL global model. Let us name this new GAN architecture *ARgan* for further reference, where an *ARgan* instance is associated with each class, as shown in Figure III.2.4.

In Algorithm 16 the lines in black represent the common path between the vanilla GAN and an *ARgan* instance, while the lines in blue are specific to vanilla GAN, and the lines

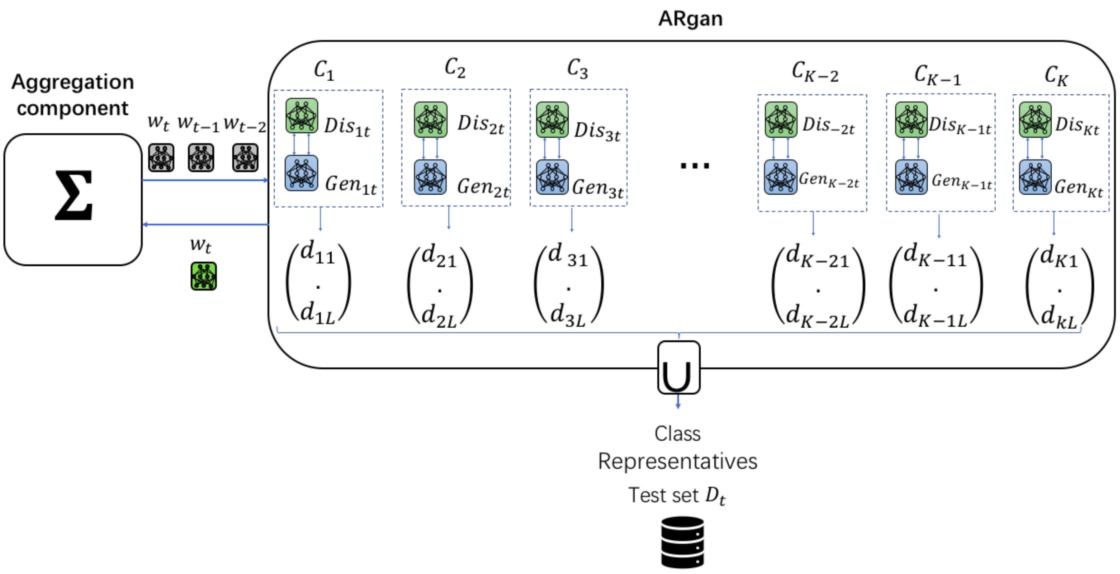

ARgan
Aggregation component
Σ
w_t w_{t-1} w_{t-2}
w_t
C_1
C_2
C_3
C_{K-2}
C_{K-1}
C_K
Dis_{1t}
Gen_{1t}
Dis_{2t}
Gen_{2t}
Dis_{3t}
Gen_{3t}
Dis_{-2t}
Gen_{K-2t}
Dis_{K-1t}
Gen_{K-1t}
Dis_{Kt}
Gen_{Kt}
...
$\begin{pmatrix} d_{11} \\ . \\ d_{1L} \end{pmatrix}$
$\begin{pmatrix} d_{21} \\ . \\ d_{2L} \end{pmatrix}$
$\begin{pmatrix} d_{31} \\ . \\ d_{3L} \end{pmatrix}$
$\begin{pmatrix} d_{K-21} \\ . \\ d_{K-2L} \end{pmatrix}$
$\begin{pmatrix} d_{K-11} \\ . \\ d_{K-1L} \end{pmatrix}$
$\begin{pmatrix} d_{K1} \\ . \\ d_{KL} \end{pmatrix}$
∪
Class
Representatives
Test set D_t

Figure III.2.4: Overview of *ARgan*

in red are specific to *ARgan*. Similar to vanilla GANs, an *ARgan* instance consists of a *generator* and a *discriminator*.

```
Algorithm 16 Vanilla GAN (blue) vs ARgan (red)
Input: Model at current round $w_t$, and class $k$
Output: Class representatives $D_t$
 1: for $e \in 1..Epochs$ do
 2:     Generate a random noise vector $X^{(t)} \leftarrow Random()$
 3:     Get fake data $F^{(t)} \leftarrow Gen_{tk}(X^{(t)})$
 4:     Feed $F^{(t)}$ to the discriminator $p^{(t)}_{fake} \leftarrow Dis_{tk}(F^{(t)})$
 5:     Compute gradients $\Delta W_{fake}$ based on $L(p^{(t)}_{fake}, C_{fake})$
 6:     Feed real data $R^{(t)}$: $p^{(t)}_{real} \leftarrow Dis_{tk}(R^{(t)})$
 7:     Compute gradients $\Delta W_{real}$ based on $L(p^{(t)}_{real}, C_{real})$
 8:     Update $Dis_{tk}$ with $\Delta W_{real} + \Delta W_{fake}$
 9:     Update $Dis_{tk}$ with $\frac{w_t}{epochs} + \Delta W_{fake}$
10:     Update $Gen_{tk}$ by computing $L(p^{(t)}, C_{real})$
11:     if $test\_accuracy(F^{(t)}, w_t) > \alpha$ then : break
12: end for
13: Output the class-representatives $D_t \leftarrow F^{(t)}$
```

The FL global model is augmented with an additional class representing fake data samples so that its gradients are compatible with the discriminator. The latter competes with the generator whose goal is to spawn data points belonging to a given label's class representatives. The adversarial training of an *ARgan* instance is similar to the five steps presented before for the vanilla GAN, except step 3 where instead of computing gradients on real data (lines 6-8), the *ARgan* instance computes artificial gradients based on the global FL model, which was trained on real data (line 9). Thus, *ARMOR* relies on $ARgan_k$ instance to generate representatives of a class C_k at every FL round t.

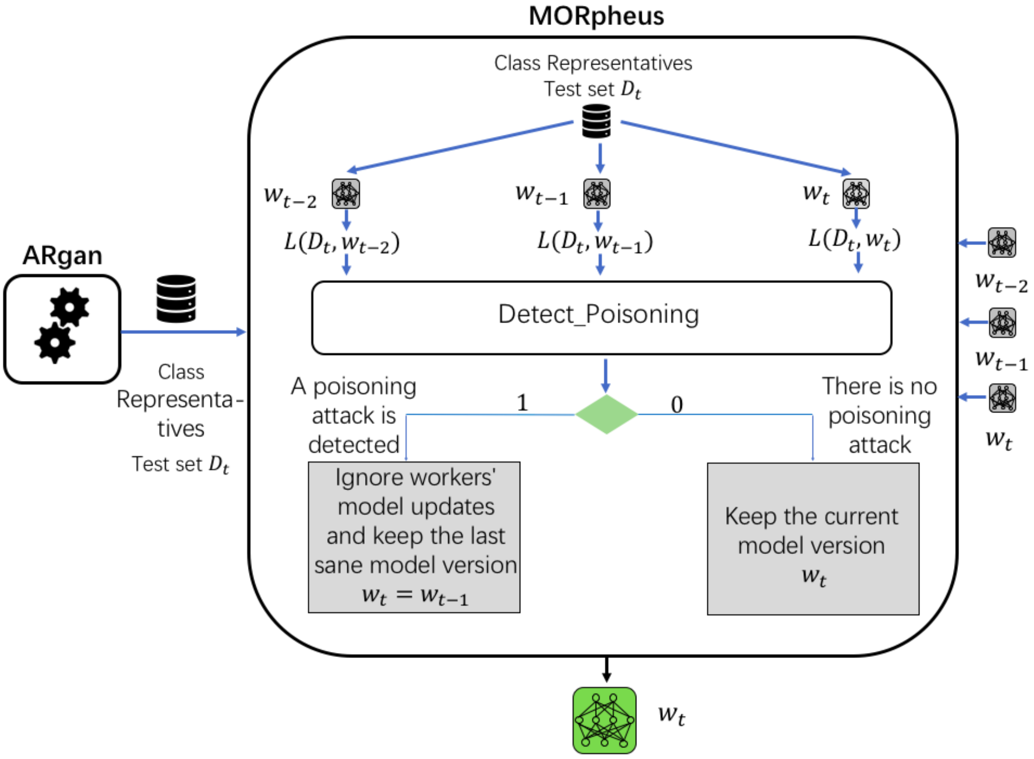


Figure III.2.5: Overview of *MORpheus*

III.2.6 *MORpheus*: *ARMOR*'s Attack Detection Mechanism

Attack detection in *ARMOR* is performed by its *MORpheus* component, which is described in Figure III.2.5. First, *MORpheus* builds a testing set D_t of class representatives for each class C_k, where $k \in [1..K]$. These class representatives are built using Algorithm 16. Afterwards, *MORpheus* feeds this testing set D_t to the current model w_t, and to the models of the two previous FL rounds w_{t-1} and w_{t-2}. Note that it is possible to use more previous model versions to detect the occurrence of poisoning based on loss monitoring. However, based on our empirical evaluation (See next chapter), we noticed that two previous models were sufficient to have good attack mitigation success rates.

The testing loss is computed for each one of these model versions and then used to detect poisoning via Eq. (III.2.1). If a poisoning is detected, workers' model updates are ignored. Otherwise, they are taken into account in the new global FL model. The detection formula is described in Eq. (III.2.1).

$$\begin{aligned} &Detect\_Poisoning\ (w_t, w_{t-1}, w_{t-2}) = \\ &\begin{cases} 1, & \text{if } (\frac{L(D_t,w_t)-L(D_t,w_{t-1})}{max(L(D_t,w_t),L(D_t,w_{t-1})} > \gamma_1 \\ & \text{and } \frac{L(D_t,w_t)-L(D_t,w_{t-2})}{max(L(D_t,w_t),L(D_t,w_{t-2})} > \gamma_2) \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{III.2.1}$$

Algorithm 17 *MORpheus*'s Attack Detection

```
    **Inputs:** Model at current round $w_t$, models at previous rounds $w_{t-1}$, and $w_{t-2}$
    **At each iteration** $t$ **do :**
    $D_t \leftarrow ClassRepresentatives(w_t)$
    Feed the class-representatives to $w_t$, $w_{i-t}$, and $w_{t-2}$
    $y_t \leftarrow w_t(D_t)$, $y_{t-1} \leftarrow w_{t-1}(D_t)$, $y_{t-2} \leftarrow w_{t-2}(D_t)$
    Compute the testing loss for $w_t$, $w_{t-1}$, and $w_{t-2}$
    $L_t \leftarrow L(y_t, k)$, $L_{t-1} \leftarrow L(y_{t-1}, k)$, $L_{t-2} \leftarrow L(y_{t-2}, k)$
    **if** $Detect\_Poisoning(w_t, w_{t-1}, w_{t-2}) == 1$ **then** :
        Ignore workers' model updates ($w_t = w_{t-1}$)
```

III.2.7 Summary

In this chapter, we presented *ARMOR*, a new mitigation mechanism against poisoning attacks that unlike state-of-the-art works relies on the informational essence of model updates to detect poisoning instead of monitoring their geometrical shapes. The intuition behind *ARMOR* is to use a set of generative-adversarial networks to generate synthetic testing data, which is used to monitor loss changes based on which poisoning is detected. In the next chapter, we present the empirical evaluation of *ARMOR* as well as its comparison with state-of-the-art works.

III.3 Evaluation of *ARMOR*

In this chapter, we present the empirical evaluation of the poisoning mitigation mechanism *ARMOR* that was presented earlier. In this evaluation, we assess the impact of this defense mechanism on model utility and its capability to counter edge-case backdoor attacks. We carry our evaluation using widely used image recognition datasets and compare our proposal to the closest state-of-the-art poisoning defense mechanisms. Our evaluation takes into consideration different settings that differ in terms of attackers' number, workers' number, training data distribution, and differential privacy noise level.

III.3.1 Implementation Details

We implemented the proposed FL defense mechanism, as well as the data poisoning and model poisoning attacks using the PyTorch framework [188]. The software prototype has 2.5 KLOC of code and is publicly available at [1] We also compare against three state-of-the-art defense mechanisms by using the implementation provided by their authors: Multi-Krum [189], NDC [190] and Trimmed Mean [191].

III.3.2 Datasets and Model Architectures

Our experiments are conducted using the real-world MNIST [192] and FashionMNIST [193] datasets for image classification tasks. These two datasets contain 50,000 training images and 10,000 test images. For the MNIST dataset, we use a five-layer neural network with three convolution layers and two fully connected layers. As for the FashionMNIST dataset, we trained a four-layer convolutional neural network with two convolution layers, a fully connected layer, and a max-pooling layer. The non-IID data distribution used in our experiments is generated using the Dirichlet distribution [194].

III.3.3 Experimental Setup

III.3.3.1 Hardware and Software Environment

All our experiments are executed on a server with 2 Intel Xeon Gold 6126 CPUs with 12 cores each, 1 Nvidia Tesla P100-PCIE-16GB GPU, with 192 GiB memory, and deployed in Ubuntu 18.04 operating system.

[1] https://gitlab.liris.cnrs.fr/rtalbi/armorfd (hidden until the corresponding research paper is published).

III.3.3.2 FL System Settings

In our experiments, we consider a total number of 10 clients, among which there is a single attacker (Unless otherwise specified). In each experiment, the models are trained for 80 FL rounds, and the adversary starts conducting its attacks from the first round. The default attack step is set to 1, where the attacker is selected and performs its data poisoning or model poisoning attack at each round. Note that the initial model used at the beginning of the experiments is pre-trained until a proper accuracy is reached.

For the aggregation algorithm used by the FL server, we use FedAvg [151], where each worker trains its model for five local epochs with a local learning rate of 0.01. We assume that all workers are always selected in each FL round. The training batch size is set to 64. The parameters of all defense mechanisms that we consider in our evaluations were empirically chosen. We use the following default parameters unless stated otherwise: $\beta = 20\%$ of dropped highest and lowest updates for Trimmed Mean, we set the number of byzantine workers to $f = 1$ for Multi-Krum, and use a norm bound of $M = 5$ for NDC. For a fair comparison, different configurations of each state-of-art defense mechanism were evaluated to choose the one with the best behavior. When it comes to *ARMOR*, we set the maximum number of *ARgan* epochs to 100, and use the following detection thresholds $\gamma_1 = 0.12$, $\gamma_2 = 0.12$, and $\alpha = 90$.

III.3.3.3 Evaluation Metrics

Backdoor task accuracy. This metric corresponds to the number of data samples of the attacker's testing set that fall into the target class C_t divided by the size of this testing set.

Model accuracy this metric corresponds to the F1-score value obtained when evaluating the accuracy of the global model's main task using a backdoor-free testing set.

Attack success per round given a threshold δ that is empirically chosen (fixed in our experiments to $\delta = 60\%$), if the backdoor task's accuracy at a given round exceeds δ, the attack success for that round is equal to 1; otherwise it is equal to 0.

Attack success rate it is the average of all values of *attack success per round*, for all training rounds between the first time an attack is injected and the last round of the experiment.

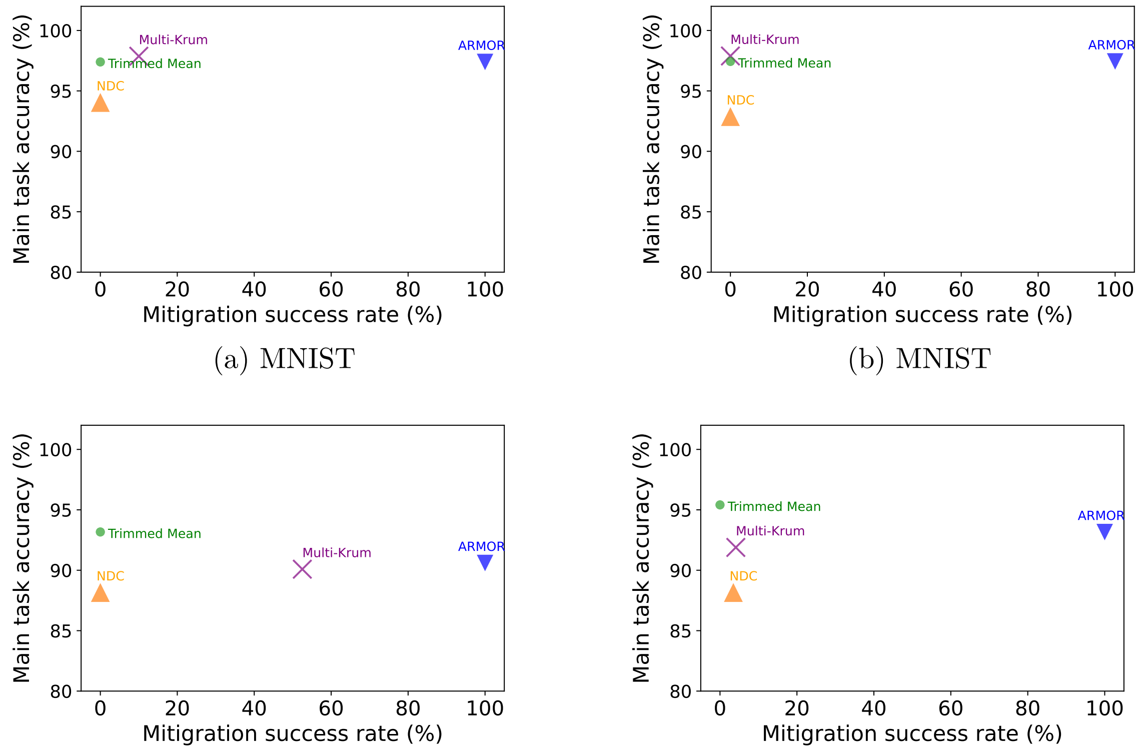
Mitigation success rate this metric is equal to $1 \ - \ \textit{attack success rate}$.

Runtime cost it measures the average execution time of an FL round when using a specific defense mechanism against poisoning attacks.

III.3.4 Experimental Results

III.3.4.1 *ARMOR* Achieving the Defense Objectives

In the following, we evaluate the trade-off between the model's accuracy and its robustness against targeted data and model poisoning attacks when using *ARMOR* as well as the three state-of-the-art defense mechanisms. The attacker follows the threat model described in Section III.2.1 and tempers with training data to inject the edge-case backdoor task described in Section III.2.1.2. As shown in Figure III.3.1, *ARMOR* achieves the optimal trade-off for both of the MNIST and FashionMNIST datasets, with a high attack mitigation success rate for data and model poisoning attacks and in the same time ensures a model accuracy that is identical to an attack-free baseline.



(a) MNIST

(b) MNIST

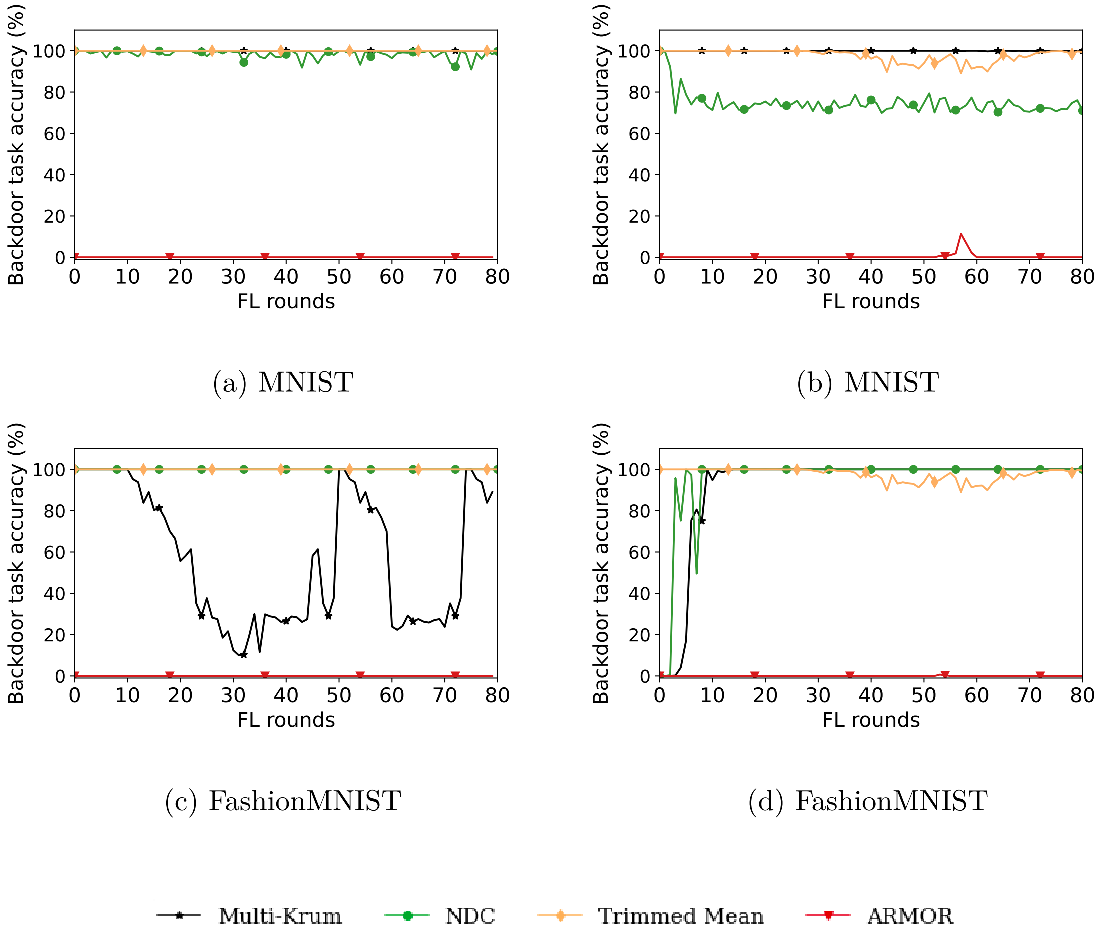
(c) FashionMNIST

(d) FashionMNIST

Figure III.3.1: Trade-off between robustness and utility – Data poisoning on the left side, and model poisoning on the right side

As for the state-of-the-art defenses, they all fail to protect against the edge-case backdoor attack in most cases, except for the Multi-Krum mechanism, that mitigates data poisoning attacks with a success rate of 4% and 52%, *c.f.*, Figures III.3.1a and III.3.1c. However, Multi-Krum fails to protect against model poisoning attacks since the adversary manages to generate model updates that are very close to the global model and are selected by its aggregation rule. Thus, compared to its competitors, *ARMOR* improves resilience to attacks by +48% to +100%.

Figure III.3.2 shows the detailed evolution of the backdoor accuracy with respect to FL rounds for the same scenarios described earlier. One can notice that *ARMOR* successfully mitigates the data and model poisoning attacks in all FL rounds for both data and model poisoning, except for round 57 in the case of model poisoning, where the backdoor task is injected only at a very low accuracy (11%). This is due to the fact that the attack at this FL round is not strong enough to provoke a loss difference that exceeds *ARMOR*'s detection thresholds. Multi-Krum has the best behavior among the state-of-the-art defense mechanisms, particularly using the FashionMNIST dataset, where the latter manages to counter data poisoning attacks at some FL rounds (between 30-50 and 60-70) and model poisoning attacks during the first 10 FL rounds. Interestingly, NDC manages to reduce the impact of model poisoning attack (with a minimal value of 70%) since this attack generates model updates with a significant norm.



(a) MNIST

(b) MNIST

(c) FashionMNIST

(d) FashionMNIST

Figure III.3.2: Target task accuracy – Data poisoning on the left side, model poisoning on the right side

Impact of Attack Frequency

In the following experiment, we evaluate the trade-off between model accuracy and backdoor accuracy for different attack frequencies using the FashionMNIST dataset, for *ARMOR*, as well as each one of the state-of-the-art poisoning defense mechanisms. We consider four different attack rates. In the first one, the attacker is selected in all FL rounds, while in the rest of the scenarios, it is selected every 2, 3, and 5 FL rounds, respectively. Figure III.3.3 illustrates the results obtained for both data (left side) and model poisoning (right side) attacks. Figure III.3.4 shows the detailed evolution of the backdoor accuracy with respect to FL rounds, considering the same attack scenarios described above. One can notice that *ARMOR* successfully mitigates the data and model poisoning attacks for all attack frequencies.

The state-of-the-art defense mechanisms either entirely or partially fail to mitigate the edge-case backdoor attacks as the attack frequency increases. For instance, Multi-Krum manages to considerably reduce the backdoor accuracy for data poisoning with an attack occurring every 5 FL rounds (Maximum accuracy of 18%). Nonetheless, for model poisoning, Multi-Krum only manages to counter the attack with the same frequency during the first 28 rounds. After that, the backdoor is successfully and sustainably injected within the model. Similar behavior is observed with higher attack frequencies, especially with model poisoning attacks where the attacker carefully crafts model updates that are close to the model of the previous round, which makes the attacker's updates more likely to be selected by Multi-Krum.

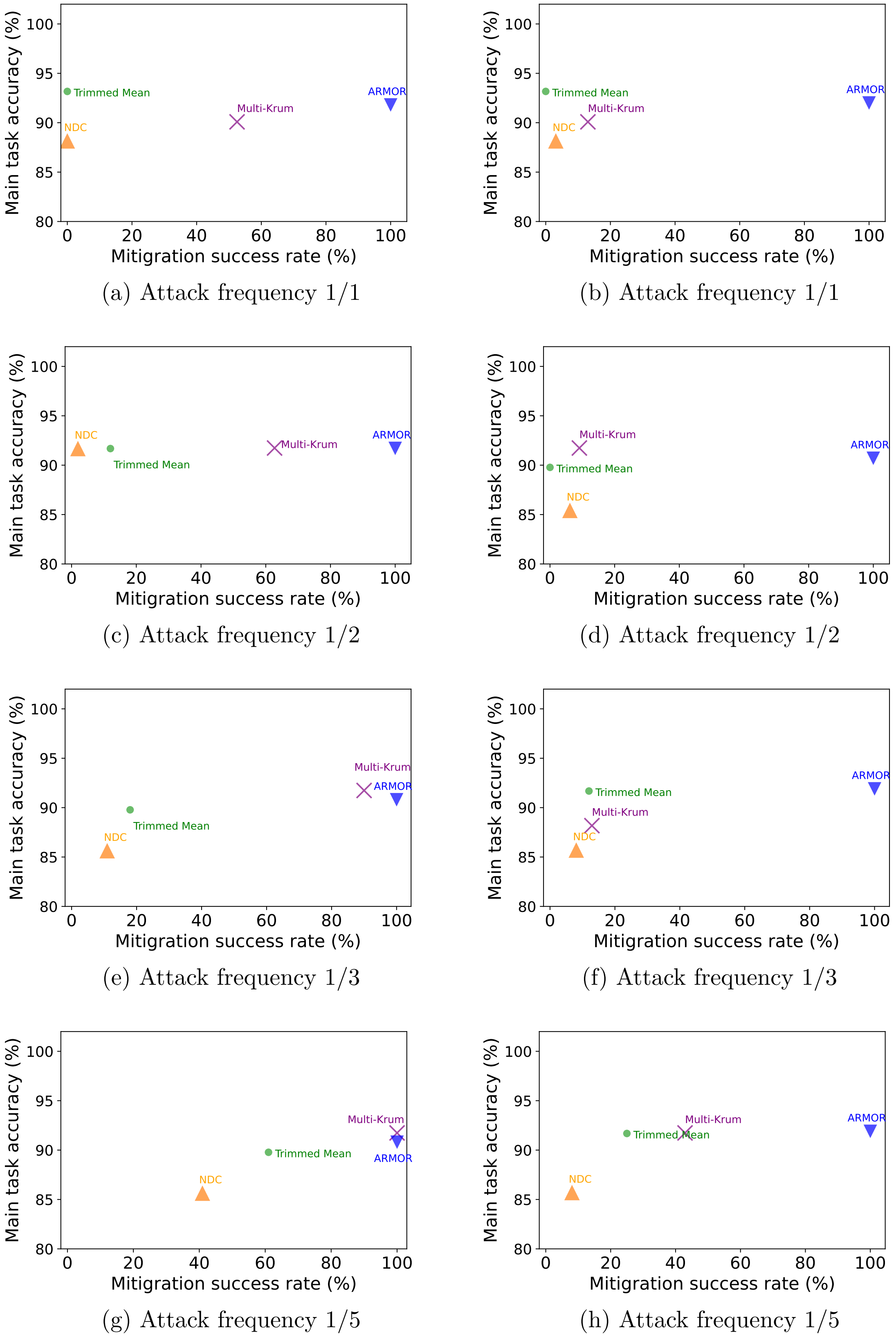
When it comes to the NDC and Trimmed Mean defense mechanisms, they either entirely fail to detect the attack or manage to counter it only during the first few rounds. Finally, the attack is always successfully injected into the model since the attacker's model updates start to become close to the global model (lower norms in the case of NDC or lower model weight values for trimmed mean). These two defense mechanisms have remarkably better performance with data poisoning and lower attack frequency because this attack is less powerful, so it takes more time to be successfully introduced within the model.

III.3.4.2 Impact of Number of Malicious Clients

Figure III.3.5 shows the impact of the number of FL malicious clients on the mitigation success rate and the accuracy of the main task, with *ARMOR* and state-of-the-art defense mechanisms applied on FashionMNIST. We assume that defense mechanisms are unaware of the variation of the number of attackers in the system. Here, the mitigation success rate metric is calculated as defined in Section III.3.3.3. We observe that *ARMOR* achieves a 100% mitigation success rate even when the number of attackers increases. *ARMOR* essentially detects the presence of the backdoor in the model, so it is orthogonal to the percentage of malicious clients.

As the number of attackers increases, the backdoor injection becomes more reliable and less detected by state-of-the-art detection mechanisms. For e. g. with Multi-Krum, the mitigation rate drops from 52% to 19% for data poisoning attacks.

For NDC, we observe that it is quite efficient in model poisoning attacks since model updates are boosted, but not very efficient with data poisoning attacks where updates have a similar norm as regular workers' updates. When we increase the number of attackers, its



(a) Attack frequency 1/1

(b) Attack frequency 1/1

(c) Attack frequency 1/2

(d) Attack frequency 1/2

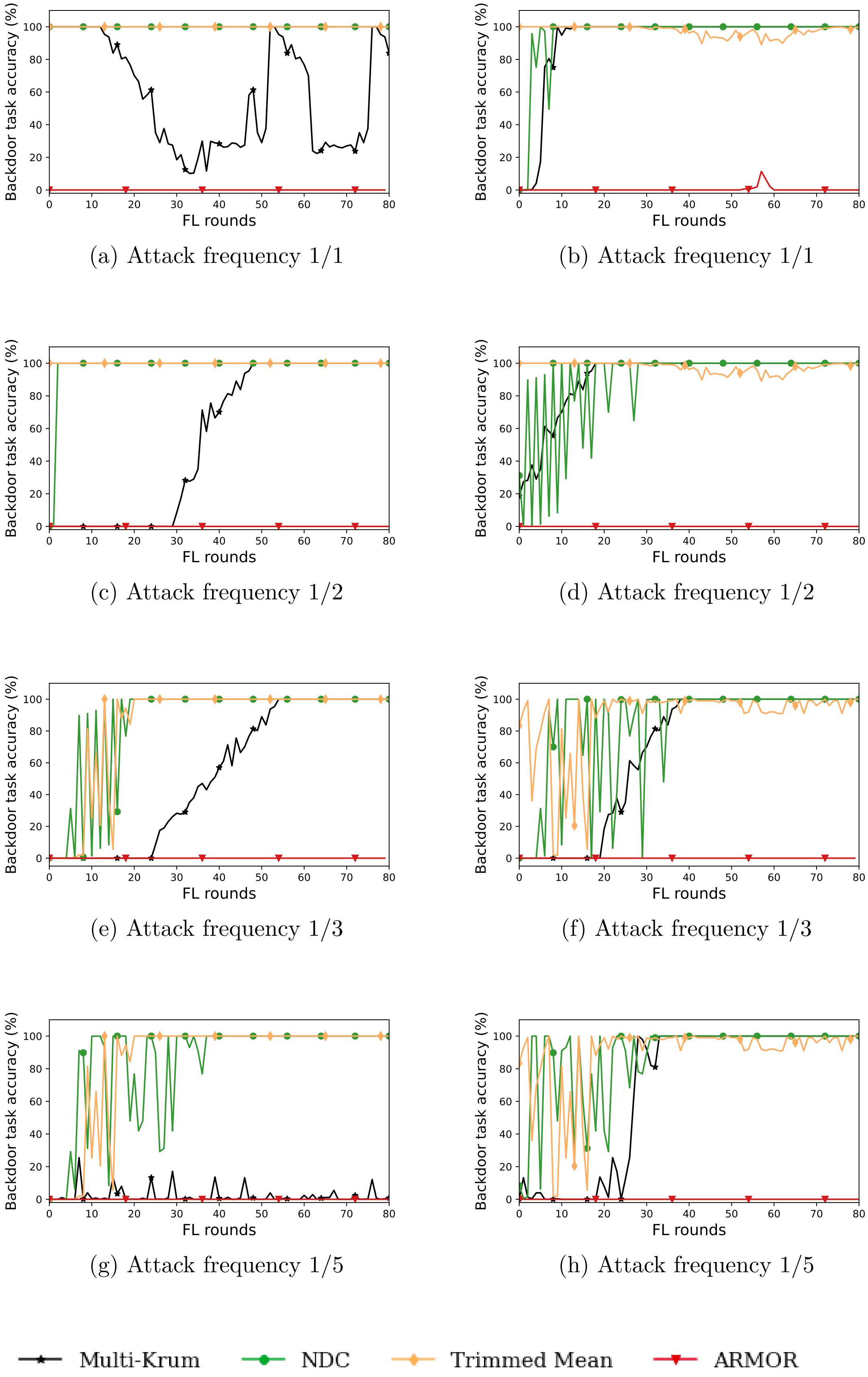
(e) Attack frequency 1/3

(f) Attack frequency 1/3

(g) Attack frequency 1/5

(h) Attack frequency 1/5

Figure III.3.3: Impact of attack frequency on robustness and utility – Data poisoning on left side, model poisoning on right side



(a) Attack frequency 1/1

(b) Attack frequency 1/1

(c) Attack frequency 1/2

(d) Attack frequency 1/2

(e) Attack frequency 1/3

(f) Attack frequency 1/3

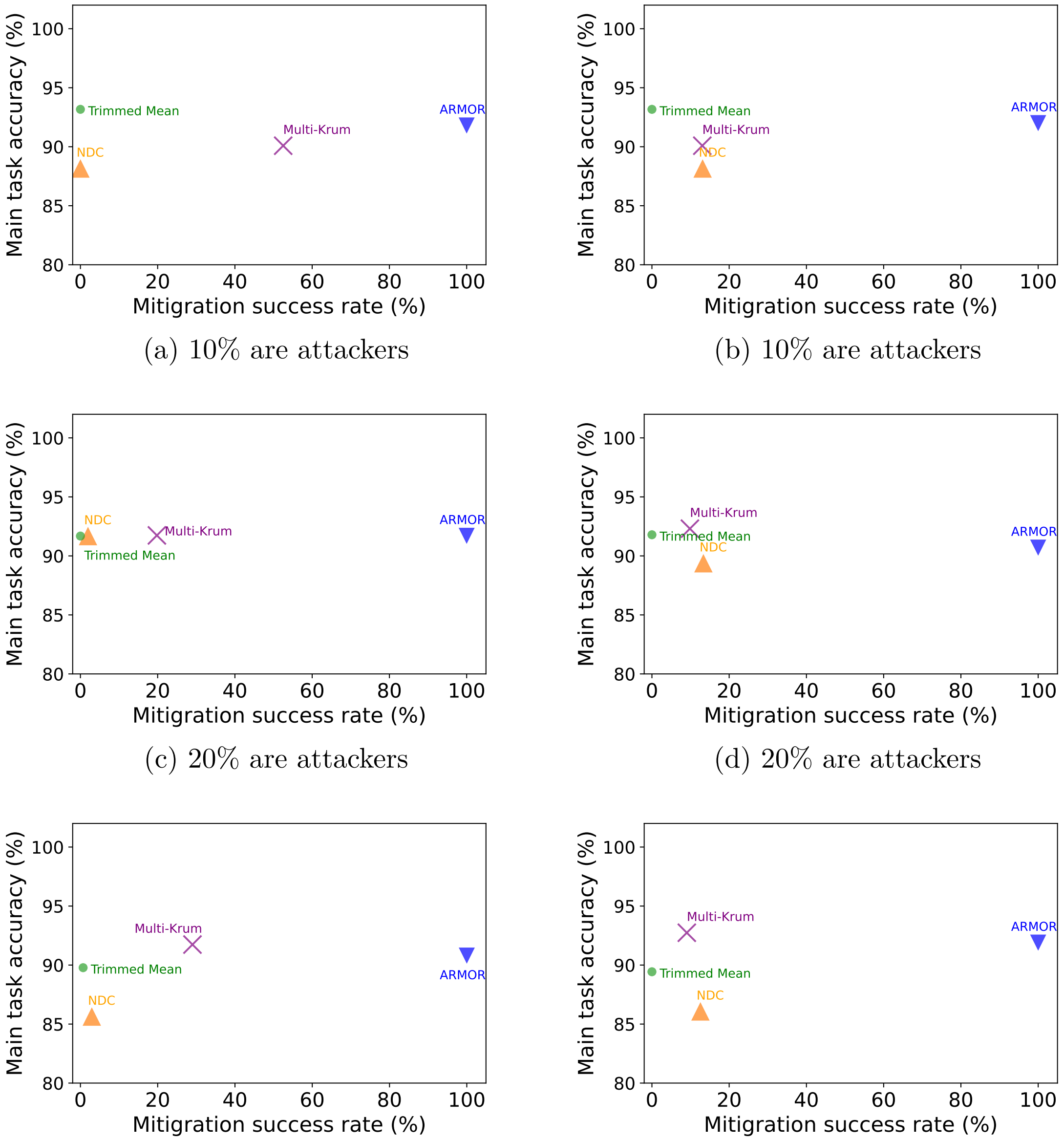
(g) Attack frequency 1/5

(h) Attack frequency 1/5

Figure III.3.4: Detailed impact of attack frequency on target task accuracy – Data poisoning on left side, model poisoning on right side

efficiency remains unchanged because even if the gradients' norm is clipped, the percentage of malicious gradients is enough to inject the backdoor.

For Trimmed Mean, the attack is never detected. However, the accuracy of the main task degrades when there are more attackers because more malicious gradients are aggregated into the global model.



(a) 10% are attackers

(b) 10% are attackers

(c) 20% are attackers

(d) 20% are attackers

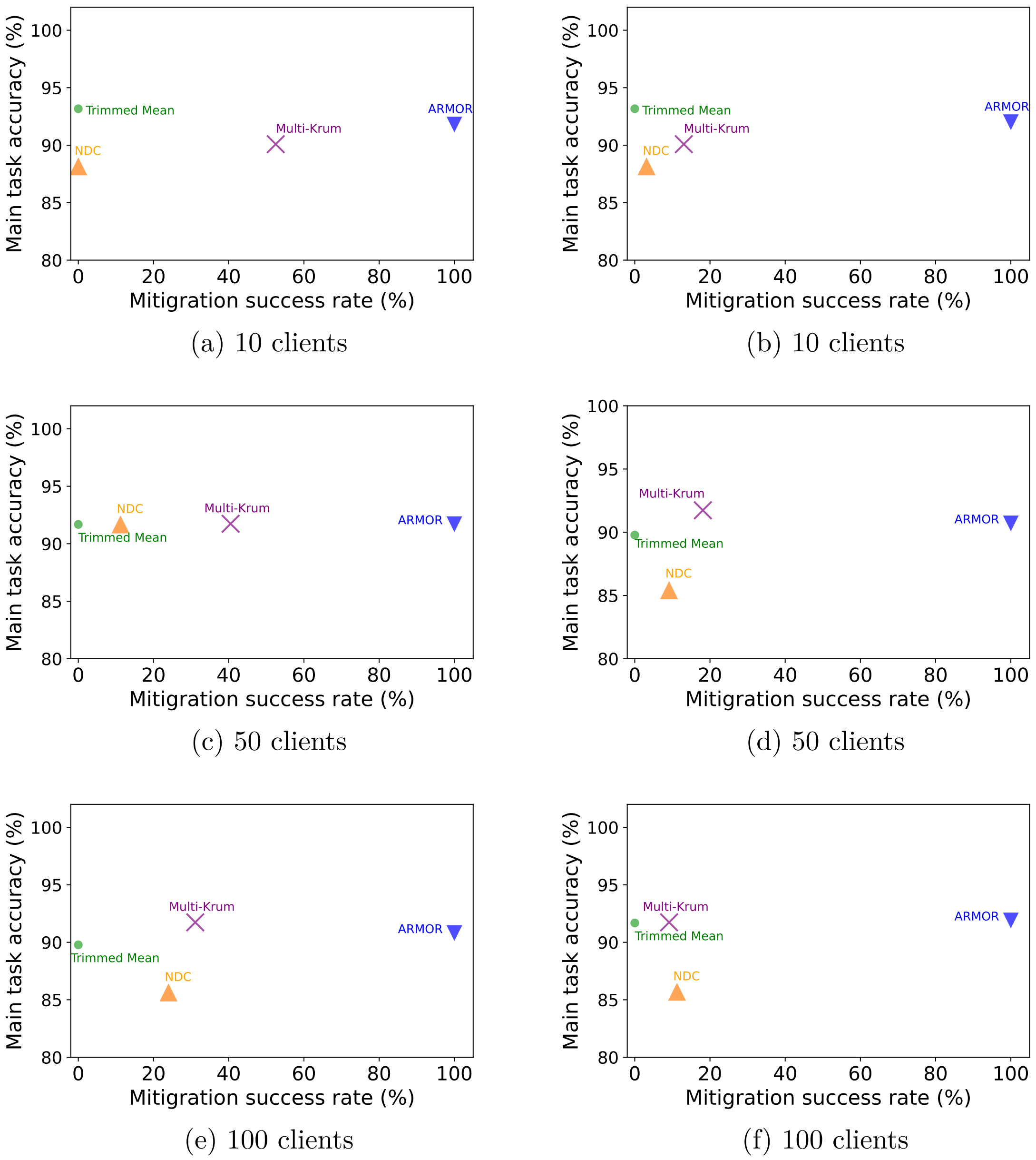
(e) 50% are attackers

(f) 50% are attackers

Figure III.3.5: Impact of attacker number on robustness and utility – Data poisoning on left side, model poisoning on right side

Impact of Total Number of Clients

Figure III.3.6 shows the mitigation success rate and accuracy with respect to the number of clients in the system, which varies from 10 to 100, keeping the percentage of attackers fixed at 10%. *ARMOR* consistently achieves a mitigation success rate of 100%. This is due to the fact that the backdoor remains the same regardless of the number of workers in the system, as the number of attackers is relative to the total number of workers in the system. The mitigation success rate for state-of-the-art systems remains essentially constant.



(a) 10 clients

(b) 10 clients

(c) 50 clients

(d) 50 clients

(e) 100 clients

(f) 100 clients

Figure III.3.6: Impact of client number on robustness and utility – Data poisoning on left side, model poisoning on right side

III.3.4.3 Effect of Non-IID Data Distributions

Figure III.3.7 shows the trade-off between the attack mitigation rate and the main task accuracy for different non-IID data distributions between different clients. These distributions were randomly generated or used the Dirichlet distribution with a parameter varying in $[0, \infty[$. The higher this parameter is, the more similar is the distribution between clients.

To evaluate *ARMOR*, we have applied three Dirichlet parameter values, namely 0.1, 0.5, and 1. The mitigation rate degrades for state-of-the-art systems like Multi-Krum and Trimmed Mean, which are only made for IID settings because with non-IID data distribution, the distance between the different updates is higher and does not necessarily indicate the presence of attackers. NDC slightly degrades the main task accuracy because the gradients come from different norms, which are reduced automatically, while *ARMOR* achieves a mitigation rate equal to 100%.

III.3.4.4 Effect of Differential Privacy

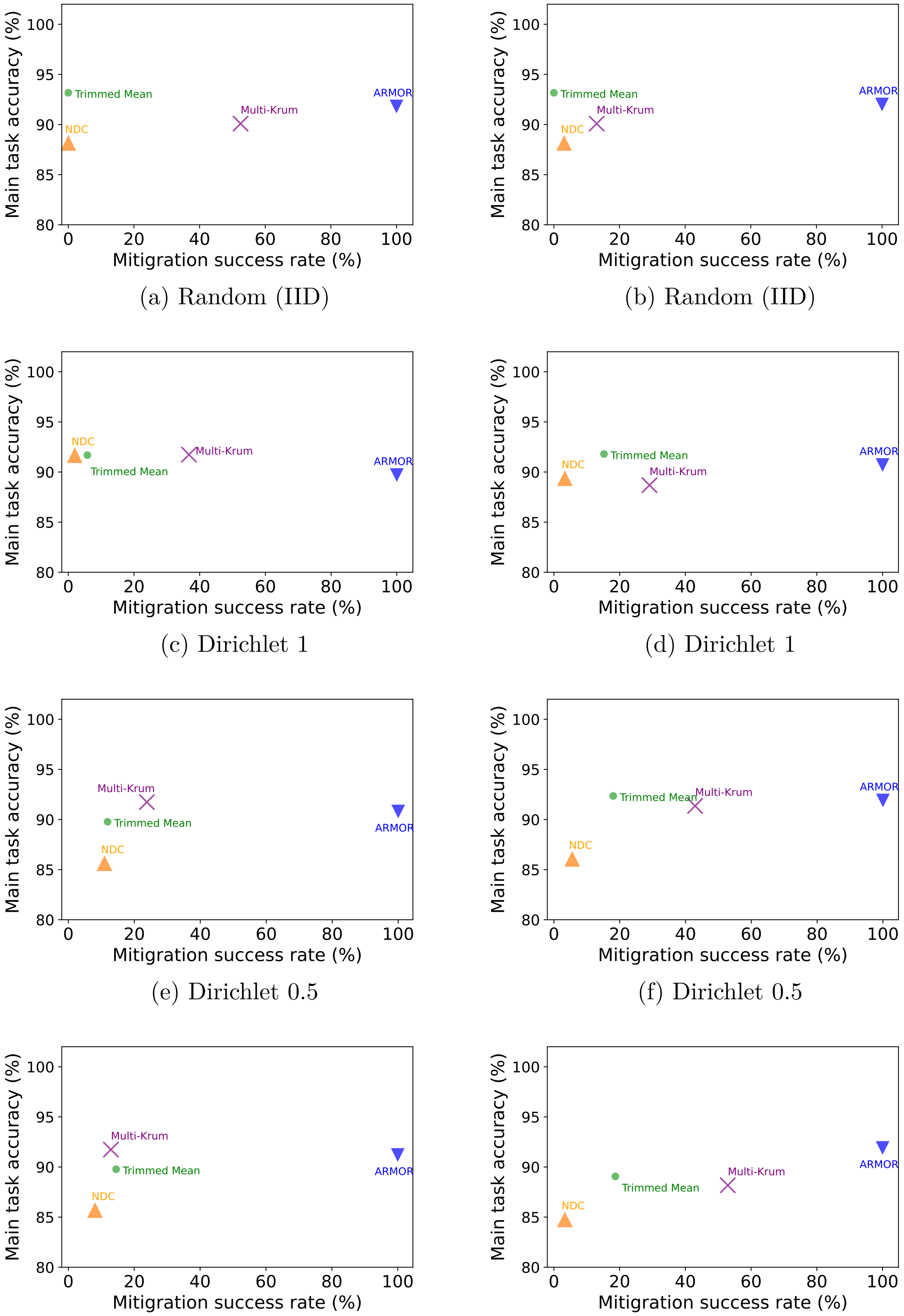
We also conducted experiments to explore the behavior of detection mechanisms when combined with differential privacy. We recall that differential privacy consists in clipping the gradients before adding a Gaussian noise configured with ϵ. Figure III.3.8 shows the tradeoff between the mitigation rate and the utility of the detectors, for different noise levels $\epsilon = \{0.5, 2, 8\}$. A low value for ϵ corresponds to a significant noise added to the updates, and conversely, a high value corresponds to a low noise. As differential privacy may perturb the attack success rate, we assume that the attacker does not add noise to its updates in our attack.

From these figures, we can see that adding differential privacy, with a medium to small noise, helps the state-of-the-art detection mechanisms mitigate the attack without hurting the overall performance. However, with a large noise, even if the mitigation rate is high, the model's utility is much lower (a drop up to 30%). ARMOR always has a higher mitigation rate than other detectors regardless of the noise magnitude.

III.3.4.5 Cost of Defense

We measured the runtime overhead induced by each one of the defense mechanisms considered in this evaluation. The results are reported in Table III.3.1. The previous experiments show that our defense mechanism *ARMOR* achieves the optimal trade-off between poisoning mitigation and model utility. However, this is done at the expense of an important runtime overhead due to the GAN training process, which has a relatively high cost. The runtime overhead of *ARMOR* is much slower than other state-of-the-art poisoning mitigation mechanisms that simply audit the geometric shape of model updates.

Nevertheless, this cost can further be reduced using parallel GPU computation on the FL server side. The per-class GANs are independent from each other, so they can be trained in parallel on multiple GPUs. Moreover, recent works aim to improve the convergence speed of GANs [195], which could greatly help to reduce *ARgan* cost.



(a) Random (IID)

(b) Random (IID)

(c) Dirichlet 1

(d) Dirichlet 1

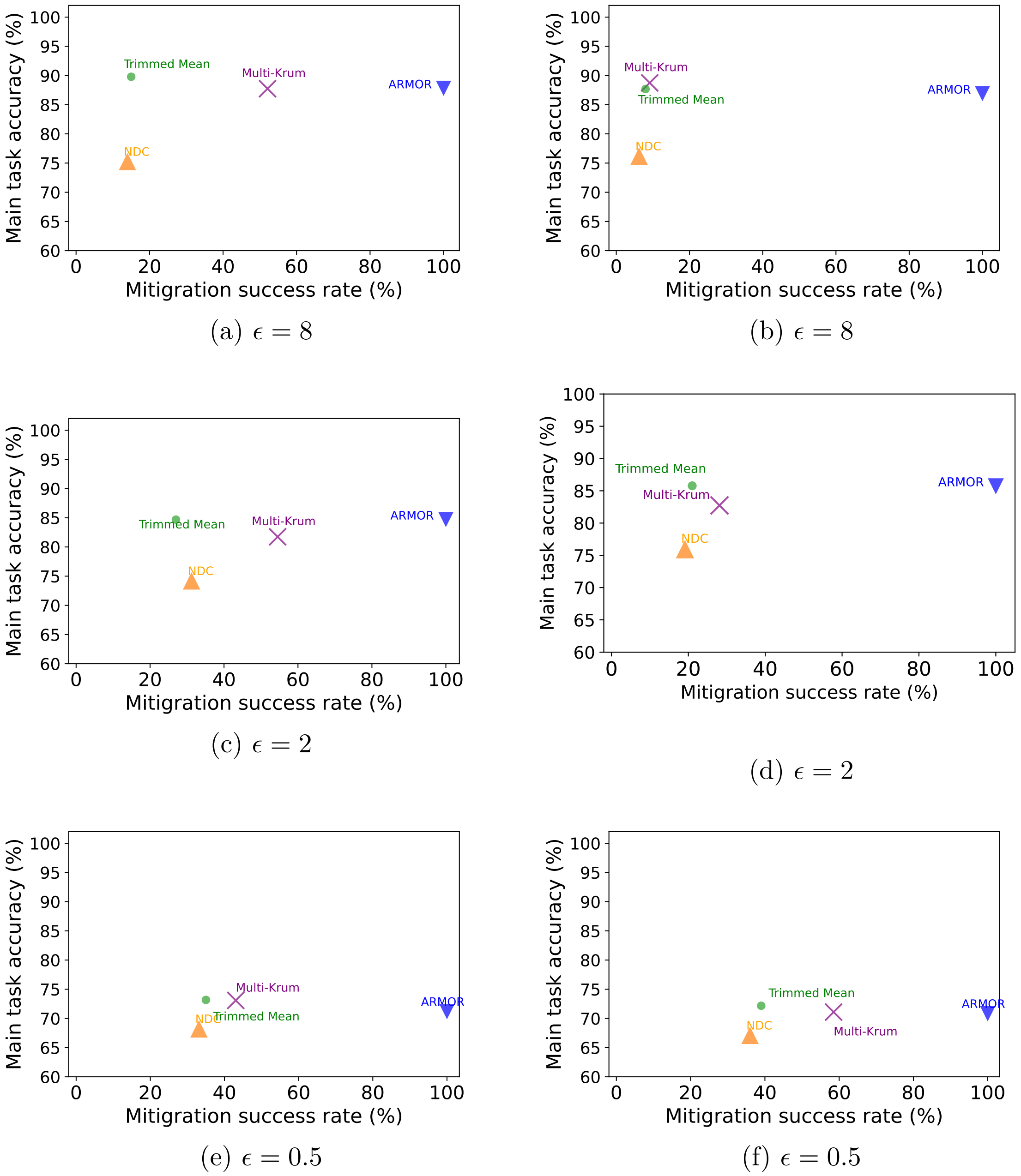
(e) Dirichlet 0.5

(f) Dirichlet 0.5

(g) Dirichlet 0.1

(h) Dirichlet 0.1

Figure III.3.7: Attack effectiveness with different non-IID settings – Data poisoning on left side, model poisoning on right side



(a) $\epsilon = 8$

(b) $\epsilon = 8$

(c) $\epsilon = 2$

(d) $\epsilon = 2$

(e) $\epsilon = 0.5$

(f) $\epsilon = 0.5$

Figure III.3.8: Attack effectiveness with various differential privacy settings – Data poisoning on left side, model poisoning on rigth side

Defense mechanism	Cost
Multi-Krum	8 ms
NDC	7 ms
Trimmed Mean	3 ms
ARMOR	21 s

Table III.3.1: Cost of defense mechanisms

III.3.5 Summary

This chapter evaluated *ARMOR*, a novel federated learning defense method against targeted poisoning attacks and, more precisely, edge-case backdoors. Our empirical evaluation was carried on two real-world datasets, MNIST and FashionMNIST, and relied on widely used neural network architectures. Our results show that *ARMOR* can achieve edge-case backdoor robustness against a significant fraction of malicious clients under different non-IID data distributions and various differential privacy settings. We demonstrate that our method outperforms *all* state-of-the-art robust FL solutions and *by large margins*. Interesting future work includes investigating techniques to improve the convergence speed of GANs in *ARgan*, and designing stronger poisoning attacks to *ARMOR*.

Part IV

Conclusion and Perspectives

Concluding Remarks

Collaborative Machine Learning is a widely used ML setting where multiple parties with common interests collaborate to get better and more accurate machine learning models using their joint data. Due to legal, financial, and competitive constraints, this scheme is not always feasible, particularly in the presence of sensitive data or when malicious participants target the system.

This thesis tackled important research questions related to the aforementioned privacy and robustness issues in collaborative machine learning frameworks and resulted in two main contributions. The first one is *PrivML*: an outsourced homomorphic encryption-based privacy preserving ML framework, while the second one is *ARMOR*: a mitigation mechanism against poisoning attacks in federated learning.

The first contribution of this thesis focused on efficiency issues in Privacy Preserving Machine Learning (PPML) that rely on cryptographic techniques, specifically Homomorphic Encryption (HE). Indeed these techniques do not impact models' utility and accuracy while providing high privacy guarantees. Still, they are known to induce high computational overheads.

In this part of the thesis, we precisely examined the impact of different architectural and design choices of HE-based PPML methods on actual PPML performance. Our foremost objective was to design and implement a privacy preserving framework that ensures a proper trade-off between privacy guarantees, computational efficiency, and service utility.

To this end, we propose *PrivML* an outsourced homomorphic encryption-based privacy preserving collaborative machine learning framework that allows optimizing runtime and bandwidth consumption for widely used ML algorithms, using many techniques such as fast algorithms for large integer arithmetic, ciphertext packing, approximate computations, and parallel computing. The software prototype of the *PrivML* framework is publically available for researchers and practitioners [1].

The second part of the thesis focused on robustness issues in a specific collaborative learning framework which is Federated Learning (FL). Unlike the outsourced setting considered in the first part of this thesis, in this framework, the different collaborators perform local training on their private data and do not outsource them to an external service provider. Instead, they only share model updates with an orchestrator who is responsible for aggregating the different model updates sent by the different collaborators, updating a global model, and communicating it to them.

This kind of scheme is designed to ensure data privacy. However, it remains vulnerable to a panoply of client-side attacks [3] including poisoning attacks [4, 5] which we studied in-depth. In these attacks, adversaries attempt to inject a malicious task into the federated model along with its main task. This malicious task assigns a label chosen by the attacker

[1] https://gitlab.liris.cnrs.fr/rtalbi/DAPPLE-2.0

to the input data with a specific trigger.

Detecting poisoning attacks in federated learning is challenging because participants only send model updates to the FL server instead of their raw training data. As a result, the FL server holds less information about users' behavior to detect malicious participants.

In this second part of the thesis, we have shown that attackers can still evade existing poisoning detectors by building model updates that mimic the updates of benign participants.

Subsequently, we proposed *ARMOR*, a new GAN-based poisoning attack detector that analyzes the information that model updates capture about users' data allowing it to mitigate extremely aggressive poisoning attack scenarios. The software prototype of *ARMOR* is also publically available for researchers and practitioners [2]

Research Perspectives

In the following, we enumerate the potential research directions of this work in both privacy preservation and robustness research questions investigated in this thesis.

Privacy Preservation in Collaborative ML

The current implementation of *PrivML* uses rigid parameter configuration regardless of the privacy requirements desired by ML service providers and users. A potential improvement of this work is to generalize it into a PPML compiler that transforms privacy and even efficiency requirements into dynamic cryptographic and ML parameter configuration. The idea is to make *PrivML* as customized as possible, where users can benefit from the optimal privacy-level and performance trade-off.

Also, although *PrivML* offers quite interesting privacy guarantees, it does not ensure protection against some inference attacks that can extract sensitive information about models or users' data only using consecutive queries. It only would be convenient to complete our work with an appropriate query auditing strategy allowing us to improve its privacy preservation guaranties.

Poisoning Mitigation in Federated Learning

In *ARMOR*, we currently use as many GAN models as the number of classes of a given dataset. Consequently, and considering how time-consuming training generative adversarial networks can be, this solution is inconvenient for datasets with more significant class numbers. A potential solution for this is to use conditional GAN to allow a single GAN to inspect poisoning for all class labels.

[2] https://gitlab.liris.cnrs.fr/rtalbi/armorfd (hidden until the corresponding research paper is published).

Another aspect that can be investigated is to use other data types to evaluate *ARMOR*. In this manuscript, we only evaluated our solution using image classification datasets which are MNIST and FashionMNIST. It would be interesting to observe how *ARMOR* behaves using more complex datasets such as Cifar100 or Trafic Signs classification datasets. It also would be interesting to consider textual data.

For now, *ARMOR* focuses on single-shot attacks since usually attackers have fewer chances of being selected all the time and tend to carry single-shot attacks that take effect immediately. However, in other contexts where somehow the attacker is sure of being selected in multiple rounds, the latter can carry adaptive attacks that accumulate the poisoning effect across multiple rounds. *ARMOR* needs to be improved to capture such attacks mainly by inspecting more previous model versions to detect the gradual poisoning.

Also, when poisoning is detected, *ARMOR* carries a rollback to the previous sane model version. However, this operation can be rather destructive since useful benign updates of that round are not exploited, and the exact identity of attacks remains undetected and thus unpunished. In improved versions of *ARMOR*, one can consider a more detailed investigation of model updates once a poisoning is detected to find the attack provenance and punish the responsible or ban them from the system.

Bibliography

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[2] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics*, 6(2):e19, 2018.

[3] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[4] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS 2020)*, pages 2938–2948, 2019.

[5] Clement Fung, Chris J M Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*, 2018.

[6] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic Regression Model Training Based on the Approximate Homomorphic Encryption. *IACR Cryptol. ePrint Arch.*, 2018:254, 2018.

[7] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):12, 2019.

[8] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[9] Dhruv Guliani, Francoise Beaufays, and Giovanni Motta. Training speech recognition models with federated learning: A quality/cost framework. *arXiv preprint arXiv:2010.15965*, 2020.

[10] Shiva Raj Pokhrel and Jinho Choi. A decentralized federated learning approach for connected autonomous vehicles. In *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 1–6. IEEE, 2020.

[11] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133*, 2020.

[12] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in byzantium. *arXiv preprint arXiv:1802.07927*, (CONF), 2018.

[13] Hian Chye Koh, Gerald Tan, et al. Data mining applications in healthcare. *Journal of healthcare information management*, 19(2):65, 2011.

[14] Vivek Bhambri. Application of data mining in banking sector. *IJCST*, 2(2):199–202, 2011.

[15] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016.

[16] Ron Kohavi. Mining e-commerce data: the good, the bad, and the ugly. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 8–13. ACM, 2001.

[17] Harvey J Miller and Jiawei Han. *Geographic data mining and knowledge discovery.* CRC Press, 2009.

[18] Pierangela Samarati. Protecting respondents identities in microdata release. *IEEE transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

[19] Ljiljana Brankovic and Vladimir Estivill-Castro. Privacy issues in knowledge discovery and data mining. In *Australian institute of computer ethics conference*, pages 89–99, 1999.

[20] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 3–18. IEEE, 2017.

[21] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz. Mlaas: Machine learning as a service. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 896–902. IEEE, 2015.

[22] Christopher M Bishop. *Pattern recognition and machine learning.* springer, 2006.

[23] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[24] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10:66–71, 2009.

[25] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[26] Muhammad Zulfadhilah, Yudi Prayudi, and Imam Riadi. Cyber profiling using log analysis and k-means clustering. *International Journal of Advanced Computer Science and Applications*, 7(7):430–435, 2016.

[27] Alan F Westin and Oscar M Ruebhausen. *Privacy and freedom*, volume 1. Atheneum New York, 1967.

[28] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, volume 29, pages 439–450. ACM, 2000.

[29] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Annual International Cryptology Conference*, pages 36–54. Springer, 2000.

[30] Vassilios S Verykios, Elisa Bertino, Igor Nai Fovino, Loredana Parasiliti Provenza, Yucel Saygin, and Yannis Theodoridis. State-of-the-art in privacy preserving data mining. *ACM Sigmod Record*, 33(1):50–57, 2004.

[31] Charu C Aggarwal and S Yu Philip. A general survey of privacy-preserving data mining models and algorithms. In *Privacy-preserving data mining*, pages 11–52. Springer, 2008.

[32] Lei Xu, Chunxiao Jiang, Jian Wang, Jian Yuan, and Yong Ren. Information security in big data: privacy and data mining. *IEEE Access*, 2:1149–1176, 2014.

[33] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[34] Kun Liu, Chris Giannella, and Hillol Kargupta. An attacker's view of distance preserving maps for privacy preserving data mining. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 297–308. Springer, 2006.

[35] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[36] Stephen E Fienberg and Julie McIntyre. Data swapping: Variations on a theme by dalenius and reiss. In *International Workshop on Privacy in Statistical Databases*, pages 14–29. Springer, 2004.

[37] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, technical report, SRI International, 1998.

[38] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *null*, page 24. IEEE, 2006.

[39] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115. IEEE, 2007.

[40] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[41] Mike Atallah, Elisa Bertino, Ahmed Elmagarmid, Mohamed Ibrahim, and Vassilios Verykios. Disclosure limitation of sensitive rules. In *Knowledge and Data Engineering Exchange, 1999.(KDEX'99) Proceedings. 1999 Workshop on*, pages 45–52. IEEE, 1999.

[42] Stanley RM Oliveira, Osmar R Zaiane, and Yücel Saygin. Secure association rule sharing. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 74–85. Springer, 2004.

[43] Yücel Saygin, Vassilios S Verykios, and Chris Clifton. Using unknowns to prevent discovery of association rules. *ACM Sigmod Record*, 30(4):45–54, 2001.

[44] LiWu Chang and Ira S Moskowitz. Parsimonious downgrading and decision trees applied to the inference problem. In *Proceedings of the 1998 workshop on New security paradigms*, pages 82–89. ACM, 1998.

[45] Francis Y Chin and Gultekin Ozsoyoglu. Auditing and inference control in statistical databases. *IEEE Transactions on Software Engineering*, (6):574–582, 1982.

[46] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[47] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[48] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[49] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[50] Josh Benaloh. Dense probabilistic encryption. In *Proceedings of the workshop on selected areas of cryptography*, pages 120–128, 1994.

[51] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 37–54. Springer, 2003.

[52] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In *International conference on the theory and applications of cryptographic techniques*, pages 308–318. Springer, 1998.

[53] Ivan Damgård and Mads Jurik. A generalisation, a simpli. cation and some applications of paillier's probabilistic public-key system. In *International Workshop on Public Key Cryptography*, pages 119–136. Springer, 2001.

[54] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[55] Michael Fellows and Neal Koblitz. Combinatorial cryptosystems galore! *Contemporary Mathematics*, 168:51–51, 1994.

[56] Françoise Levy-dit Vehel and Ludovic Perret. A polly cracker system based on satisfiability. In *Coding, Cryptography and Combinatorics*, pages 177–192. Springer, 2004.

[57] Le Van Ly. Polly two: a new algebraic polynomial-based public-key scheme. *Applicable Algebra in Engineering, Communication and Computing*, 17:267–283, 2006.

[58] Rainer Steinwandt. A ciphertext-only attack on polly two. *Applicable Algebra in Engineering, Communication and Computing*, 21(2):85–92, 2010.

[59] Françoise Levy-dit Vehel, Maria Grazia Marinari, Ludovic Perret, and Carlo Traverso. A survey on polly cracker systems. In *Gröbner Bases, Coding, and Cryptography*, pages 285–305. Springer, 2009.

[60] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography Conference*, pages 325–341. Springer, 2005.

[61] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for nc/sup 1. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 554–566. IEEE, 1999.

[62] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography Conference*, pages 575–594. Springer, 2007.

[63] Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, pages 420–443. Springer, 2010.

[64] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010.

[65] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[66] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.

[67] Zvika Brakerski and Vinod Vaikuntanathan. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[68] Nick Howgrave-Graham. Approximate integer common divisors. In *International Cryptography and Lattices Conference*, pages 51–66. Springer, 2001.

[69] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[70] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation.* Cambridge University Press, 2015.

[71] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In *Theory of Cryptography Conference*, pages 336–354. Springer, 2004.

[72] Michael O Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[73] Joe Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31. ACM, 1988.

[74] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[75] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[76] George Robert Blakley. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*, pages 313–313. IEEE Computer Society, 1979.

[77] Edward Waring. Vii. problems concerning interpolations. *Philosophical transactions of the royal society of London*, (69):59–67, 1779.

[78] Amos Beimel. Secret-sharing schemes: a survey. In *International Conference on Coding and Cryptology*, pages 11–46. Springer, 2011.

[79] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[80] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21. Springer, 2012.

[81] Ximeng Liu, Rongxing Lu, Jianfeng Ma, Le Chen, and Baodong Qin. Privacy-preserving patient-centric clinical decision support system on naive bayesian classification. *IEEE journal of biomedical and health informatics*, 20(2):655–668, 2016.

[82] Ximeng Liu, Robert H Deng, Kim-Kwang Raymond Choo, and Jian Weng. An efficient privacy-preserving outsourced calculation toolkit with multiple keys. *IEEE Transactions on Information Forensics and Security*, 11(11):2401–2414, 2016.

[83] Sangwook Kim, Masahiro Omori, Takuya Hayashi, Toshiaki Omori, Lihua Wang, and Seiichi Ozawa. Privacy-Preserving Naive Bayes Classification Using Fully Homomorphic Encryption. In *International Conference on Neural Information Processing*, pages 349–358. Springer, 2018.

[84] XiaoFeng Wang, Haixu Tang, Shuang Wang, Xiaoqian Jiang, Wenhao Wang, Diyue Bu, Lei Wang, Yicheng Jiang, and Chenghong Wang. idash secure genome analysis competition 2017, 2018.

[85] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[86] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11(4):83, 2018.

[87] RL Kennedy, HS Fraser, LN McStay, and RF Harrison. Early diagnosis of acute myocardial infarction using clinical and electrocardiographic data at presentation: derivation and evaluation of logistic regression models. *European heart journal*, 17(8):1181–1191, 1996.

[88] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Logistic regression on homomorphic encrypted data at scale. 33(01):9466–9471, 2019.

[89] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[90] Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *IMA International Conference on Cryptography and Coding*, pages 45–64. Springer, 2013.

[91] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[92] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[93] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, pages 483–512. Springer, 2018.

[94] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33. Springer, 2016.

[95] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321. ACM, 2015.

[96] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2018.

[97] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[98] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[99] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 1:24, 2019.

[100] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, 2016.

[101] Nikhil Ketkar. Introduction to pytorch. In *Deep learning with python*, pages 195–208. Springer, 2017.

[102] Morten Dahl, Jason Mancuso, Yann Dupis, Ben Decoste, Morgan Giraud, Ian Livingstone, Justin Patriquin, and Gavin Uhma. Private machine learning in tensorflow using secure computation. *arXiv preprint arXiv:1810.08130*, 2018.

[103] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[104] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

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

[106] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. Ezpc: Programmable, efficient, and scalable secure two-party computation

for machine learning. Technical report, Cryptology ePrint Archive, Report 2017/1109. https://eprint. iacr. org/2017/1109, 2017.

[107] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[108] John C Mitchell, Rahul Sharma, Deian Stefan, and Joe Zimmerman. Information-flow control for programming on encrypted data. In *2012 IEEE 25th Computer Security Foundations Symposium*, pages 45–60. IEEE, 2012.

[109] Dahlia Malkhi, Noam Nisan, Benny Pinkas, Yaron Sella, et al. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, volume 4, page 9. San Diego, CA, USA, 2004.

[110] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

[111] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. Oblivm: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy*, pages 359–376. IEEE, 2015.

[112] Janus Dam Nielsen and Michael I Schwartzbach. A domain-specific programming language for secure multiparty computation. In *Proceedings of the 2007 workshop on Programming languages and analysis for security*, pages 21–30. ACM, 2007.

[113] Aseem Rastogi, Matthew A Hammer, and Michael Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *2014 IEEE Symposium on Security and Privacy*, pages 655–670. IEEE, 2014.

[114] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348. IEEE, 2013.

[115] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*, volume 821. John Wiley & Sons, 2012.

[116] Hyeong-Jin Kim, Hyeong-Il Kim, and Jae-Woo Chang. A privacy-preserving knn classification algorithm using yao's garbled circuit on cloud computing. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 766–769. IEEE, 2017.

[117] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.

[118] Paul Viola, Michael Jones, et al. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1:511–518, 2001.

[119] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631. ACM, 2017.

[120] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1651–1669. USENIX Association, 2018.

[121] Yehuda Lindell. How to
te it–A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.

[122] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Kdd*, volume 2, page 4, 2000.

[123] David Maxwell Chickering, Christopher Meek, and Robert Rounthwaite. Efficient Determination of Dynamic Split Points in a Decision Tree. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 91–98. IEEE, 2001.

[124] Lichun Li, Rongxing Lu, Kim-Kwang Raymond Choo, Anwitaman Datta, and Jun Shao. Privacy-preserving-outsourced association rule mining on vertically partitioned databases. *IEEE Transactions on Information Forensics and Security*, 11(8):1847–1861, 2016.

[125] Pei-Yih Ting and Xiao-Wei Huang. Distributed Paillier Plaintext Equivalence Test. *IJ Network Security*, 6(3):258–264, 2008.

[126] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[127] Christine Jost, Ha Lam, Alexander Maximov, and Ben JM Smeets. Encryption performance improvements of the paillier cryptosystem. *IACR Cryptol. ePrint Arch.*, 2015:864, 2015.

[128] Pierrick Gaudry, Alexander Kruppa, and Paul Zimmermann. A gmp-based implementation of schönhage-strassen's large integer multiplication algorithm. In *Proceedings of the 2007in1982auditing international symposium on Symbolic and algebraic computation*, pages 167–174, 2007.

[129] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.5 edition, 2012. `http://gmplib.org/`.

[130] Ran Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.

[131] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 3.0, May 2008.

[132] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A Multiple-Precision Binary Floating-Point Library with Correct Rounding. *ACM Transactions on Mathematical Software (TOMS)*, 33(2):13, 2007.

[133] Ximeng Liu, Robert Deng, Kim-Kwang Raymond Choo, and Yang Yang. Privacy-Preserving Outsourced Support Vector Machine Design for Secure Drug Discovery. *IEEE Transactions on Cloud Computing*, 2018.

[134] Chen Li and Wenping Ma. Comments on "An Efficient Privacy-Preserving Outsourced Calculation Toolkit with Multiple Keys". *IEEE Transactions on Information Forensics and Security*, 13(10):2668–2669, 2018.

[135] Dheeru Dua and Casey Graff. UCI machine learning repository, https://archive.ics.uci.edu/ml, 2017.

[136] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[137] The Cambridge Analytica Scandal. `https://www.theguardian.com/news/series/cambridge-analytica-files`.

[138] T.C. Sottek and Janus Kopfstein. Everything You Need to Know About PRISM. `https://www.theverge.com/2013/7/17/4517480/nsa-spying-prism-surveillance-cheat-sheet`.

[139] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *MLSys*, 2019.

[140] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

[141] Guodong Long, Yue Tan, Jing Jiang, and Chengqi Zhang. Federated learning for open banking. In *Federated learning*, pages 240–254. Springer, 2020.

[142] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *29th USENIX Security Symposium (USENIX Security 2020)*, pages 1605–1622, 2020.

[143] Virat Shejwalkar and Amir Houmansadr. Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning. In *Network and Distributed Systems Security Symposium (NDSS 2021)*, 2021.

[144] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing Federated Learning Through an Adversarial Lens. In *International Conference on Machine Learning (ICML 2019)*, pages 634–643, 2019.

[145] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. *arXiv preprint arXiv:2007.05084*, 2020.

[146] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[147] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger. Braintorrent: A peer-to-peer environment for decentralized federated learning. *arXiv preprint arXiv:1905.06731*, 2019.

[148] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, pages 125–136, 2020.

[149] Wenyu Zhang, Xiumin Wang, Pan Zhou, Weiwei Wu, and Xinglin Zhang. Client selection for federated learning with non-iid data in mobile edge computing. *IEEE Access*, 9:24462–24474, 2021.

[150] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[151] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2017.

[152] Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence, 2016.

[153] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers, 2019.

[154] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging, 2020.

[155] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *European Symposium on Research in Computer Security*, pages 480–501, 2020.

[156] Di Cao, Shan Chang, Zhijian Lin, Guohua Liu, and Donghong Sun. Understanding distributed poisoning attack in federated learning. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 233–239. IEEE, 2019.

[157] Yang Liu, Zhihao Yi, and Tianjian Chen. Backdoor attacks and defenses in feature-partitioned collaborative learning. *arXiv preprint arXiv:2007.03608*, 2020.

[158] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. *arXiv preprint arXiv:1911.11815*, 2019.

[159] Chien-Lun Chen, Leana Golubchik, and Marco Paolieri. Backdoor attacks on federated meta-learning. *arXiv preprint arXiv:2006.07026*, 2020.

[160] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. Learning to detect malicious clients for robust federated learning. *arXiv preprint arXiv:2002.00211*, 2020.

[161] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

[162] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753. IEEE, 2019.

[163] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pages 14774–14784, 2019.

[164] Wenqi Wei, Ling Liu, Margaret Loper, Ka-Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. A framework for evaluating gradient leakage attacks in federated learning. *arXiv preprint arXiv:2004.10397*, 2020.

[165] Jierui Lin, Min Du, and Jian Liu. Free-riders in federated learning: Attacks and defenses. *arXiv preprint arXiv:1911.12560*, 2019.

[166] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[167] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[168] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[169] James Newsome, Brad Karp, and Dawn Song. Paragraph: Thwarting signature learning by training maliciously. In *International Workshop on Recent Advances in Intrusion Detection*, pages 81–105. Springer, 2006.

[170] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles Sutton, J Doug Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. *LEET*, 8:1–9, 2008.

[171] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J Doug Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 1–14, 2009.

[172] Andrew Newell, Rahul Potharaju, Luojie Xiang, and Cristina Nita-Rotaru. On the practicality of integrity attacks on document-level sentiment analysis. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 83–93, 2014.

[173] Mehran Mozaffari-Kermani, Susmita Sur-Kolay, Anand Raghunathan, and Niraj K Jha. Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE journal of biomedical and health informatics*, 19(6):1893–1905, 2014.

[174] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[175] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[176] Peva Blanchard, Rachid Guerraoui, Julien Stainer, and Others. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, pages 119–129, 2017.

[177] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Generalized Byzantine-Tolerant SGD. *arXiv e-prints*, page arXiv:1802.10116, February 2018.

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

[179] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

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

[181] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. Brendan McMahan. Can you really backdoor federated learning? *CoRR*, abs/1911.07963, 2019.

[182] Krishna Pillutla, Sham M. Kakade, and Zaid Harchaoui. Robust aggregation for federated learning, 2019.

[183] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. FLTrust: Byzantine-Robust Federated Learning via Trust Bootstrapping. In *Network and Distributed Systems Security Symposium (NDSS 2021)*, 2021.

[184] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. The Limitations of Federated Learning in Sybil Settings. 2020.

[185] Marco Barreno, Blaine Nelson, Anthony D Joseph, and J Doug Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[186] Gilad Baruch, Moran Baruch, and Yoav Goldberg. A Little Is Enough: Circumventing Defenses For Distributed Learning. In *Advances in Neural Information Processing Systems (NeurIPS 2019)*, volume 32, 2019.

[187] Lie He, Sai Praneeth Karimireddy, and Martin Jaggi. Byzantine-Robust Learning on Heterogeneous Datasets via Resampling. *arXiv e-prints*, page arXiv:2006.09365, 2020.

[188] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIP 2019)*, pages 8024–8035. 2019.

[189] Sébastien Rouault Georgios Damaskinos, Arsany Guirguis. Aggregathor. `https://github.com/LPD-EPFL/AggregaThor`, 2019.

[190] Hongyi Wang Kartik Sreenivasan. Ood federated learning. `https://github.com/ksreenivasan/OOD-Federated-Learning`, 2020.

[191] Shuhao Fu, Chulin Xie, Bo Li, and Qifeng Chen. Attack-resistant federated learning with residual-based reweighting. `https://github.com/fushuhao6/Attack-Resistant-Federated-Learning`, 2019.

[192] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[193] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

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

[195] Jiachen Zhong, Xuanqing Liu, and Cho-Jui Hsieh. Improving the speed and quality of gan by adversarial training. *arXiv preprint arXiv:2008.03364*, 2020.

FOLIO ADMINISTRATIF

THESE DE L'UNIVERSITE DE LYON OPEREE AU SEIN DE L'INSA LYON

NOM : TALBI
(avec précision du nom de jeune fille, le cas échéant)

DATE de SOUTENANCE : 19/11/2021

Prénoms : Rania

TITRE: Robust and Privacy Preserving Distributed Machine Learning

NATURE : Doctorat

Numéro d'ordre : 2021LYSEI077

Ecole doctorale : InfosMaths

Spécialité : Informatique

RESUME :

Avec l'omniprésence des services numériques, d'´énormes quantités de données sont continuellement générées et collectées. Les algorithmes d'apprentissage automatique (ML) permettant d'extraire des connaissances précieuses à partir de ces données et ont été appliqués dans de nombreux domaines, tels que l'assistance médicale, le transport, la prédiction du comportement des utilisateurs, et bien d'autres.
Dans beaucoup de ces applications, les données sont collectées à partir de différentes sources et un entraînement distribué est nécessaire pour apprendre des modèles globaux sur ces données. Néanmoins, dans le cas de données sensibles, l'exécution d'algorithmes ML traditionnels sur ces données peut conduire à de graves violations de la vie privée en divulguant des informations sensibles sur les propriétaires et les utilisateurs des données.

Dans cette thèse, nous proposons des mécanismes permettant d'améliorer la préservation de la vie privée et la robustesse dans le domaine de l'apprentissage automatique distribué.

La première contribution de cette thèse s'inscrit dans la catégorie d'apprentissage automatique respectueux de la vie privée basé sur la cryptographie.

De nombreux travaux de l'état de l'art proposent des solutions basées sur la cryptographie pour assurer la préservation de la vie privée dans l'apprentissage automatique distribué. Néanmoins, ces travaux sont connus pour induire d'énormes coûts en termes de temps d'exécution et d'espace. Dans cette lignée de travaux, nous proposons PrivML, un framework externalisé d'apprentissage collaboratif basé sur le chiffrement homomorphe, qui permet d'optimiser le temps d'exécution et la consommation de bande passante pour les algorithmes ML les plus utilisés, moyennant de nombreuses techniques telles que le packing, les calculs approximatifs et le calcul parallèle.

Les autres contributions de cette thèse abordent les questions de robustesse dans le domaine de l'apprentissage fédéré.

En effet, l'apprentissage fédéré est le premier framework à garantir la préservation de la vie privée par conception dans le cadre de l'apprentissage automatique distribué. Néanmoins, il a été démontré que ce framework est toujours vulnérable à de nombreuses attaques, parmi lesquelles nous trouvons les attaques par empoisonnement, où les participants utilisent délibérément des données d'entraînement erronées pour provoquer une mauvaise classification au moment de l'inférence.

Nous démontrons que les mécanismes de mitigation de l'empoisonnement de l'état de l'art ne parviennent pas à détecter certaines attaques par empoisonnement et nous proposons ARMOR, un mécanisme de mitigation de l'empoisonnement pour l'apprentissage fédéré qui parvient à détecter ces attaques sans nuire à l'utilité des modèles.

MOTS-CLÉS :
Préservation de la vie privée, robustesse, apprentissage automatique distribué, apprentissage fédéré, attaques par empoisonnement, chiffrement homomorphe.

Laboratoire (s) de recherche :

LIRIS

Directeur de thèse:

Pr. Sara Bouchenak

Président de jury :

Composition du jury :

- Benjamin Nguyen, Professeur INSA Val de Loire, Rapporteur.
- Marc Tommasi Professeur à l'université de Lille, Rapporteur.
- Lydia Chen, Maître de conférences, TU Delft, Examinatrice.
- Lionel Brunie, Professeur INSA Lyon, Examinateur.
- Sara Bouchenak, Professeur INSA Lyon, Directrice de thèse.