Privacy-preserving Machine Learning Techniques

Dissertation

submitted to

Sorbonne Université and EURECOM

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Author:

Beyza BOZDEMİR

Thesis Supervisor: **Melek ÖNEN**

Scheduled for defense on December 10th 2021, in front of a committee composed of:

Reviewers

Dr.	**Sébastien CANARD**	Orange Innovation, France
Prof.	**Katerina MITROKOTSA**	University of St.Gallen, Switzerland

Examiners

Prof.	**Zekeriya ERKIN**	Delft University of Technology, Netherlands
Dr.	**Laurent GOMEZ**	SAP Security Research, France
Prof.	**Refik MOLVA**	EURECOM, France
Prof.	**Thorsten STRUFE**	Karlsruhe Institute of Technology, Germany

Thesis Supervisor

Prof.	**Melek ÖNEN**	EURECOM, France

Apprentissage automatique et Confidentialité des données

Thèse

soumise à

Sorbonne Université et EURECOM

pour l'obtention du Grade de Docteur

présentée par:

Beyza BOZDEMİR

Directeur de Thèse: **Melek ÖNEN**

Soutenance de thèse prévue le 10 Décembre 2021 devant le jury composé de:

Rapporteur

Dr.	**Sébastien CANARD**	Orange Innovation, France
Prof.	**Katerina MITROKOTSA**	Université de St.Gallen, Suisse

Examinateur

Prof.	**Zekeriya ERKIN**	Université Technologique de Delft, Pays-Bas
Dr.	**Laurent GOMEZ**	Recherche Sécurité de SAP, France
Prof.	**Refik MOLVA**	EURECOM, France
Prof.	**Thorsten STRUFE**	Institut Technologique de Karlsruhe, Allemagne

Directeur de Thèse

Prof.	**Melek ÖNEN**	EURECOM, France

İsoş'uma

Abstract

Machine Learning as a Service (MLaaS) refers to a service that enables companies to delegate their machine learning tasks to single or multiple untrusted but powerful third parties, namely cloud servers. Thanks to MLaaS, the need for computational resources and domain expertise required to execute machine learning techniques is significantly reduced. Nevertheless, companies face increasing challenges with ensuring data privacy guarantees and compliance with the data protection regulations. Executing machine learning tasks over sensitive data requires the design of privacy-preserving protocols for machine learning techniques.

In this thesis, we aim to design such protocols for MLaaS and study three machine learning techniques: Neural network classification, trajectory clustering, and data aggregation under privacy protection. In our solutions, our goal is to guarantee data privacy while keeping an acceptable level of performance and accuracy/quality evaluation when executing the privacy-preserving variants of these machine learning techniques. In order to ensure data privacy, we employ several advanced cryptographic techniques: Secure two-party computation, homomorphic encryption, homomorphic proxy re-encryption, multi-key homomorphic encryption, and threshold homomorphic encryption. We have implemented our privacy-preserving protocols and studied the trade-off between privacy, efficiency, and accuracy/quality evaluation for each of them.

Résumé

L'apprentissage automatique en tant que service (MLaaS) fait référence à un service qui permet aux entreprises de déléguer leurs tâches d'apprentissage automatique à un ou plusieurs serveurs puissants, à savoir des serveurs cloud. Néanmoins, les entreprises sont confrontées à des défis importants pour garantir la confidentialité des données et le respect des réglementations en matière de protection des données. L'exécution de tâches d'apprentissage automatique sur des données sensibles nécessite la conception de nouveaux protocoles garantissant la confidentialité des données pour les techniques d'apprentissage automatique.

Dans cette thèse, nous visons à concevoir de tels protocoles pour MLaaS et étudions trois techniques d'apprentissage automatique : les réseaux de neurones, le partitionnement de trajectoires et l'agrégation de données. Dans nos solutions, notre objectif est de garantir la confidentialité des données tout en fournissant un niveau acceptable de performance et d'utilité. Afin de préserver la confidentialité des données, nous utilisons plusieurs techniques cryptographiques avancées : le calcul bipartite sécurisé, le chiffrement homomorphe, le rechiffrement proxy homomorphe ainsi que le chiffrement à seuil et le chiffrement à clé multiples. Nous avons en outre implémenté ces nouveaux protocoles et étudié le compromis entre confidentialité, performance et utilité/qualité pour chacun d'entre eux.

Acknowledgements

Thanks a lot for whom reading my work which reflects my deep, enthusiastic feelings for cryptography and its applications of real-world use cases.

I would like to firstly thank my lovely husband İsoş'um for his endless love, support, understanding, and time that we discussed many ideas during this study. Your convincing skills always make me interested in new things, even I sometimes questionnaire myself, courageous and successful at almost anything more than I imagine. I would not have written this thesis if I had not had the luck of being with you.

My dear big two families, I would like to thank each member for the endless love, understanding, and support during my whole life.

I would like to express my great appreciation to my thesis supervisor Melek Önen for her patient guidance (when I am stubborn :)), enthusiastic encouragement/support, valuable advice, and let me free and responsible during the development and preparation of this thesis. Her willingness to share her time and experiences with me has brightened my path.

I thank all my friends Tuğba Akbaş, Sercan Alıcı, Betül Aşkın Özdemir, Sevdenur Baloğlu, Leyla Bilge, Ayşe Candan, Cüneyt Çalışkan, Sarra Ghdifi, Birgül Koç, Nurcan and Aziz Memiş, Rumi, Merve Öğünç, Merve Şahin, Gamze Tillem, and Zeynep Ünal for friendship, sharing their valuable time with me, flavoring to my life, and their guiding during my undergraduate, master, and PhD studies. I also thank all my friends and everyone at EURECOM, especially my office mates Dimitrios, Savino, Giovanni, Paul, Simone, Pierre, and Luigi, people from Digital Security, namely Oubaida, Chiara, Khawla, Pepe, Hemlata, Mohamed, Wanying, Madhu, Alessandro, Pox, Dario, all Andrea's, Matteo, and Afiqah, and people from Communication System and Data Science, namely Sihem, Sabra, Roya, Tsu-han, Mohit, Cedric ROUX, Mounia, Robert, Harald, Omid, Ehsan, all Ali's, Mohammed, Naser, Panos, Sofia, Berksan, Türker, Onur, Chien-Chun, and Eunjeong. I also thank people I met and played beach volley during summer times.

A more than special thanks go to all the jury members in my PhD committee, namely Sébastien Canard, Zeki Erkin, Laurent Gomez, Katerina Mitrokotsa, Refik Molva, and Thorsten Strufe, for having accepted to join in the jury, questions during my defense, and valuable compliments on my work.

My research has influenced my research by my co-authors, Gamze, Mohamad, Orhan, Helen, Thomas, Sébastien, Monir, Federica, Alessandro, Jakub, and Melek. Big thanks go to all people who have been part of my publications.

Special thanks to Refik Hoca, Nick, Max, Yannick, Aurélien, and Antonio for their

support and the chat, especially during lunch.

Finally, I would like to thank our partners from the PAPAYA Project, Marco and Sauro from MCI Italy, Boris and Ron IBM Haifa, Simone and Tobias from KAU, Sébastien and Monir from Orange, and Angel from ATOS Spain. I remember our GA meeting, beautiful places I visited, and social dinners' chats.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

The acronyms and abbreviations used throughout the manuscript are specified in the following. They are presented here in their singular form, and their plural forms are constructed by adding and *s*, e.g. TX (transmitter) and TXs (transmitters). The meaning of an acronym is also indicated the first time that it is used. The English acronyms are also used for the French summary.

2PC	Secure Two-party Computation
ABY	Arithmetic-Boolean-Yao's sharings
Act	Activation Layer
Agg	Aggregator
AS	Arithmetic Sharing
BN	Batch Normalisation
BS	Boolean Sharing
CM	Confusion Matrix
CNN	Convolutional Neural Network
Conv	Convolutional Layer
CS	Cloud Server
D	Dataset
DA	Data Analyser
DBCV	Density-Based Clustering Validation
DBSCAN	Density Based Spatial Clustering of Applications with Noise
DO	Data Owner
ECG	Electro-Cardiogram
ELU	Exponential Linear Units
FC	Fully Connected Layer
FHE	Fully Homomorphic Encryption
GDPR	General Data Protection Regulation
HE	Homomorphic Encryption
H-PRE	Homomorphic Proxy Re-Encryption
IoT	Internet of Things
LHE	Levelled Homomorphic Encryption
LWE	Learning with Errors
MK-FHE	Multi-key Fully Homomorphic Encryption
M	A Neural Network model

MDL	Minimum Description Length
ML	Machine Learning
MLaaS	Machine Learning as a Service
MP	Model Provider
MPC	Secure Multi-party Computation
NN	Neural Network
NSA	The US National Security Agency
OT	Oblivious Transfer
Q_i	Querier i
PCA	Principle Component Analysis
PHE	Partially Homomorphic Encryption
Pool	Pooling Layer
PPT	Probabilistic Polynomial-Time
pp-TRACLUS	Privacy-preserving TRACLUS
PRE	Proxy Re-Encryption
PReLU	Parametric ReLU
ReLU	Rectified Linear Units
RLWE	Ring Learning with Errors
SC	Silhouette Coefficient
SC_{noise}	Silhouette Coefficient with noise
SGD	Stochastic Gradient Descent
SIMD	Single Instruction Multiple Data
SwHE	Somewhat Homomorphic Encryption
Th-FHE	Threshold Fully Homomorphic Encryption
TTP	Trusted Third Party
TRACLUS	TRAjectory CLUStering
X	Input data to NN
Y	Output/Result data of NN

List of Publications

We have presented the following publications during this PhD study. Most of them have already been presented through different venues enumerated below:

Publications – Conference

1. Monir Azraoui, Muhammad Bahram, **Beyza Bozdemir**, Sébastien Canard, Eleonora Ciceri, Orhan Ermis, Ramy Masalha, Marco Mosconi, Melek Önen, Marie Paindavoine, Boris Rozenberg, Bastien Vialla, and Sauro Vicini, "SoK: Cryptography for Neural Network", In IFIP Summer School on Privacy and Identity Management, 18-23 August 2019, Brugg Windisch, Switzerland, August 2019.

2. Mohamad Mansouri, **Beyza Bozdemir**, Melek Önen, and Orhan Ermis, "PAC: Privacy-preserving arrhythmia classification with neural networks", In FPS 2019, 12th International Symposium on Foundations and Practice of Security, November 5-7, 2019, Toulouse, France, November 2019. **Best paper award**.

3. Gamze Tillem, **Beyza Bozdemir**, and Melek Önen, "SwaNN: Switching among cryptographic tools for privacy-preserving neural network predictions", In SECRYPT 2020, 17th International Conference on Security and Cryptography, 8-10 July 2020, Lieusaint-Paris, France, Lieusaint - Paris, France, July 2020.

4. **Beyza Bozdemir**, Orhan Ermis, and Melek Önen, "ProteiNN: Privacy-preserving one-to-many Neural Network classifications, In SECRYPT 2020, 17th International Conference on Security and Cryptography, 8-10 July 2020, Lieusaint-Paris, France, Lieusaint - Paris, France, July 2020.

5. **Beyza Bozdemir**, Sébastien Canard, Orhan Ermis, Helen Möllering, Melek Önen, and Thomas Schneider, "Privacy-preserving Density-based Clustering", In ACM ASIACCS 2021, 16th ACM ASIA Conference on Computer and Communications Security, Hong Kong, China, June 2021.

6. **Beyza Bozdemir**, Federica Germinario, Alessandro Pisani, Jakub Klemsa, and Melek Önen. "PRIDA:PRIvacy-preserving data aggregation with multiple Data Analysers". Under submisson, September 2021.

Publications – Poster

1. Gamze Tillem, **Beyza Bozdemir**, and Melek Önen, "Private neural network predictions", In ICT.OPEN2019, Dutch Digital Conference, March 19-20, 2019, Hilversum, Netherlands, March 2019.

2. **Beyza Bozdemir**, Gamze Tillem, Melek Önen, and Orhan Ermis, "Privacy preserving neural network classification: A hybrid solution", In PUT 2019, Open Day for Privacy, Usability, and Transparency, co-located with the 19th Privacy Enhancing Technologies Symposium, July 15, 2019, KTH, Stockholm, Sweden, July 2019. **Best poster award**.

Chapter 1

Introduction

The starting point of all achievement is desire.

Napoleon Hill

With the recent advances in the information technology, Internet of Things (IoT) devices are found everywhere: At our homes, on our wrists or in our pockets, and the development of user-friendly applications for these IoT devices have encouraged people for their extensive usage and large amounts of data production. With the evolution of cloud computing technologies, data-driven businesses, e.g., companies or service providers, easily *collect and store* massive amount of data. Such *abundance of data* allows deriving relevant information about their users through advanced analytics such as *statistical analysis* (sum, average, etc.) or *machine learning techniques* (neural networks, clustering, etc.). These analytical findings can help companies improve their existing customer services or offer new ones. Companies are also attracted to share these data.

1.1 Machine Learning as a Service

The cloud computing technology and the success of machine learning techniques lead to a paradigm shift in technological services that enable data-driven companies to delegate their machine learning tasks to cloud servers that have domain-specific expertise in machine learning and computational resources for the required analytics. One such service is called *Machine Learning as a Service* (MLaaS) [1] illustrated in Figure 1.1.

MLaaS enables companies to easily and quickly get started with machine learning techniques: Companies do not need to install several software systems to run the required technique.

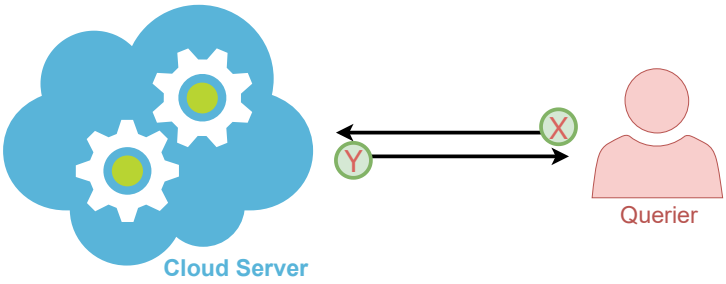


Figure 1.1 – Machine Learning as a Service

1.2 Data Privacy vs Machine Learning Techniques

The privacy of an individual is one of his/her fundamental rights. Launching MLaaS over data coming from individuals has become more and more attractive for data-driven companies with the increasing ability of data processing (collect, perform some advanced data analytics over the data, etc.). Yet, collected, stored, processed, or shared data are usually privacy-sensitive data, and, in recent years, several data protection regulations such as the European General Data Protection Regulation (GDPR) [2] or the ePrivacy directive [3] have been emerged to protect individuals' privacy. These data protection regulations guarantee that the processed data to be protected and not to be disclosed the privacy of any person. The *privacy-by-design* approach of the GDPR can be applied by employing cryptographic techniques to support data protection and, at the same time, the use of machine learning techniques over these protected data.

Machine learning techniques can be considered as a system designed to learn or solve problems based on its environment observations [4]. Today, several machine learning techniques are well-known. In this thesis, we focus on three techniques:

1. *Neural networks* are inspired from the human brain composed of many connected neurons which serve some functionality for the human body [5,6]. A neural network consists of two phases: A training phase (or called learning phase), where the neural network model is built by learning/gaining new capabilities from data in the training dataset; and a classification phase (or called prediction or querying phase), in which the created neural network model is tested with new data from the testing dataset (i.e., different from the data items in the training dataset). The neural network model can be defined as the composition of several functions taking input matrices and/or vectors that are built during the training phase: These matrices and vectors are created by determining the influence of a given neuron input on the output of that neuron. The classification phase uses the built model by taking testing data and outputting a label over them. A neural network generally consists of three different layers: Input layer, hidden layer(s), and output layer. For hidden layers, the most used ones are the convolutional layer (if image classification is needed), fully connected layer, activation layer, and pooling layer. These layers can be considered as functions such as matrix-vector multiplication, the `Max` computation, or the computation of `Sigmoid`.

2. Clustering is a machine learning technique that allows the grouping of similar data items in the same cluster. In particular, we focus on trajectory clustering, namely *TRACLUS* which is a density-based clustering technique designed and optimised for clustering trajectories [7]. TRACLUS consists of two phases: A partitioning phase, in which trajectories are divided into sub-trajectories, namely line segments (represented by two points) as close to the original trajectories as possible; and a grouping phase (or called clustering phase), where segmented sub-trajectories are grouped into some clusters according to their similarities, i.e., being neighbours of some line segment(s). These phases of TRACLUS include several functions such as the logarithm computation, the sine function, or division.

3. *Data Aggregation* is the process of regrouping data, presenting them in a summarised form, and performing some statistical analysis such as a simple sum or mean computation over these data. The underlying data usually come from multiple data sources (who can be individuals or several data-driven companies' data pools) to bring them together. Data aggregation is one of the most used data processes in the area of finance (retail, investment, etc.), the travel industry, sensor networks, or search engines [8].

The *collected/processed/shared data* paradigm raises *serious privacy concerns* mainly because of *the high sensitivity of the data*. When companies try to bring value out of them, they face increasing challenges with *ensuring data privacy guarantees* and compliance with the data protection regulations [2, 3]. Moreover, the collection, processing, and sharing of the underlying data may cause breach of individuals' privacy, e.g., leaking the date, place, or participation of some social event. There exist recent examples of privacy breaches originating from these kinds of application usage: According to two news of the Guardian, (i) some documents provided by Edward Snowden in 2014 revealed that NSA had used the mobile game Angry Birds to collect users' data such as age, gender, and location[1]; and (ii) the Facebook-Cambridge Analytica data scandal in early 2018 hit the headline when it was revealed that the consulting company had harvested the Facebook data to profile US voters and used this information to have an impact on the US voters' decision[2]. As these kinds of personal information leakage affect a country's election result, companies' future can be put in danger: They may suffer from reputation damage, operational downtime, or financial loss (e.g., the data breaches cost is globally £3.2M over five years[3]).

The private data should be confidential and private when companies employ machine learning techniques (or as a task of MLaaS on some cloud platform) over them; however, these techniques cannot work well without having an access to the underlying data. Therefore, in order to keep the data confidential and, at the same time, to enable these techniques to work properly over these privacy-sensitive data, one can leverage

[1] https://www.theguardian.com/world/2014/jan/27/nsa-gchq-smartphone-app-angry-birds-personal-data

[2] https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election

[3] https://www.metacompliance.com/blog/5-damaging-consequences-of-a-data-breach/

advanced cryptographic techniques such as homomorphic encryption or secure multi-party computation.

1.3 Privacy-preserving protocols for Machine Learning Techniques

In order to design a privacy-preserving protocol for the previously mentioned machine learning techniques, one should identify the main privacy requirements.

While developing a privacy-preserving protocol, we foresee the following requirements:

(i) ***Input privacy***: Provided input should be confidential to only its actual owner.

(ii) ***Output privacy***: The cloud server is not allowed to learn the output/result of the ML techniques' evaluation.

(iii) ***Model privacy***: A private ML model which is also an asset should not be disclosed to any party, except its owner.

(iv) The querier/data owner and the cloud server are semi-honest.

Performing machine learning techniques over confidential data usually requires the use of cryptographic techniques such as homomorphic encryption (HE) or secure multi(two)-party computation (MPC/2PC); however, these, unfortunately, incur a non-negligible overhead with respect to computational and communication costs. To efficiently combine the underlying cryptographic techniques with machine learning techniques, the design of the latter needs to be revisited. The goal of privacy-preserving machine learning protocols is to address the trade-off between privacy, efficiency, and accuracy/quality evaluation. Therefore, we identify four challenges when designing privacy-preserving protocols for neural network, (trajectory) clustering, and data aggregation:

- *Challenge 1: Complex operations.* Existing cryptographic techniques can be incompatible with some complex machine learning operations such as the `Sigmoid` activation function in the neural network.

- *Challenge 2: Optimisation of the underlying technique.* A neural network contains two phases: The training phase and the classification phase. Similarly, TRACLUS involves the partitioning phase and the clustering (or grouping) phase. The training (or partitioning) phase is composed of several iterations of the same functions' process until having a good level of accuracy (a good level of clustering quality). The training phase (or the partitioning) phase also consists of more complex operations. While comparing it with the classification (or clustering) phase, the training (or partitioning) phase performed by the cloud server requires the tuning of neural network (or TRACLUS partitioning) parameters or the increase of the number of activation functions. Therefore, such a need implies several interactions between the data owner and the cloud server. When considering the integration of this phase with cryptographic techniques, the underlying cryptographic technique cannot support them easily. We, thus, focus on the grouping (or called clustering)

phase and assume that the data owner has already run the partitioning phase over plaintext trajectories and divided them into line segments.

Furthermore, a neural network is composed of multiple layers, multiple neurons in these layers, and input and output vectors. These parameters, which define the size of the neural network, have a significant impact on the complexity of the neural network, and therefore, such parameters should be optimised while their integration with cryptographic techniques.

Moreover, (trajectory) clustering algorithms usually have some specific parameters like the number of clusters or the number of iterations. Such parameters have a non-negligible impact on the complexity of the required clustering technique, and further, the clustering quality evaluation. When designing its privacy-preserving variant, these parameters should be chosen carefully.

– *Challenge 3: Real numbers.* Neural networks or trajectory clustering computes several functions over real numbers whereas cryptographic techniques work over binary or integer numbers.

– *Challenge 4: Multiple data sources.* Another challenge is the number of data sources. The data collected from multiple sources raise data privacy challenges due to the multiplicity of data sources, each of them requiring its own privacy, and their data, therefore, should be protected individually.

We propose to study the privacy requirements with respect to two scenarios:

1. *Privacy-preserving single-server machine learning techniques:*

   Consider a scenario that involves two parties: *A querier (or a data owner)* who has a private input, and *a cloud server*, which is untrusted but powerful and owns/receives a private ML model (or the cloud server owns/receives nothing, but performs the required ML technique). The querier/data owner wishes to outsource the evaluation of some MLaaS task to the cloud server. The first requirement to achieve in this scenario is to guarantee *input privacy* against any unauthorised parties during the processing lifetime of the data (i.e., from its collection to its analysis). The unauthorised party can be the cloud server(s), querier(s)/data owner(s), or an external party, who does not play any role during the protocol. The second requirement is to ensure *output privacy* against any unauthorised parties since the output of the required machine learning techniques over privacy-sensitive input data can reveal some information about the query. Lastly, when the machine learning technique is the neural network, *the neural network model* might be *confidential* against any party except its owner since the underlying model can disclose some privacy-sensitive information about the training data, and therefore, this can indicate the identity of an individual. Note that in some cases, the model should be kept private against even the cloud server itself.

2. *Privacy-preserving two-server machine learning techniques*

A querier or data owner outsources its privacy-sensitive input data to two non-colluding semi-honest cloud servers, which execute the privacy-preserving machine learning technique tasks over the underlying data. The querier wishes to hide its sensitive input data from both cloud servers as well as the corresponding privacy-sensitive output. Moreover, when both cloud servers own/receive a private ML model, they keep the underlying model confidential. We aim to decrease the computational and communication burden of the MLaaS task of the querier and delegate the expensive operations to these two untrusted but powerful cloud servers. The previously enumerated privacy requirements should be ensured by this scenario as well.

When designing privacy-preserving variants of machine learning techniques, one should consider the security requirements, challenges or limitations in terms of computational and communication costs for a querying party, namely the querier or the data owner, and the computing party who is the cloud server. Furthermore, the complex operations in the requested machine learning technique are needed to be approximated into some linear functions if they cannot be easily compatible with cryptographic techniques.

1.4 Contributions

This thesis focuses on data confidentiality while executing privacy-preserving variants of neural networks, trajectory clustering, and data aggregation as well as maintaining their quality evaluation and efficiency. Also, we investigate the suitability of cryptographic techniques to neural networks, trajectory clustering, and data aggregation, more specifically the utilisation of secure two-party computation, homomorphic encryption, and homomorphic proxy re-encryption. We develop privacy-preserving protocols for these machine learning techniques by addressing the previously identified challenges. We divide this thesis into two parts:

In the first part of this thesis, we describe the design, development, and implementation of privacy-preserving machine learning techniques, namely neural networks and trajectory clustering, in the scenario involving a single cloud server.

1. PAC is a solution for designing privacy-preserving neural network classification for heart arrhythmia that keeps queriers' heartbeat data confidential against the cloud server and the neural network model confidential against the queriers. As a case study, we have designed a new model based on the PhysioBank dataset[4]. This model was built from scratch following the privacy-by-design approach in order to be compatible with secure two-party computation (2PC) (See Section 4.4).

2. SwaNN is a privacy-preserving neural network classification combining the additively homomorphic Paillier encryption scheme [9] with 2PC. Thanks to the use of the Paillier encryption algorithm for linear operations and also the x^2 activation function, the solution achieves better computational cost compared to existing

[4]https://www.physionet.org/physiobank/database/mitdb/

(Fully) Homomorphic Encryption ((F)HE)-based solutions. Different computation optimisations have been implemented (See Section 4.5).

3. ProteiNN is a privacy-preserving neural network classification solution based on the use of Homomorphic Proxy Re-Encryption (H-PRE) and additive encryption for achieving data confidentiality for the model(s), the inputs, and the corresponding results. Additionally, in ProteiNN, the model provider also has control over the model outsourced to the cloud server, and we investigate collusions between the ProteiNN players (See Section 4.6).

4. pp-TRACLUS is the first privacy-preserving trajectory clustering solution based on 2PC. We have designed an efficient protocol for trajectory clustering, in particular TRACLUS, and applied it on several real-world datasets such as a Travel dataset consisting of data over movements of people (See Section 5.4).

In the second part of this thesis, we investigate a scenario involving two non-colluding cloud servers (named as the two-server scenario) which help the client(s) or data owner(s) execute privacy-preserving neural network classification, trajectory clustering, and data aggregation tasks without gaining or disclosing any information regarding the processed data or its output. Thanks to these two cloud servers, the querier/data owner only performs minimal operations. We present four solutions in this part, namely:

1. A second version of SwaNN which can be executed in case the querier lacks resources (See Chapter 6).

2. pp-TRACLUS is a privacy-preserving trajectory clustering solution that combines the additively homomorphic Paillier encryption scheme [9] with TRACLUS. A second version of pp-TRACLUS based on 2PC is proposed since the Paillier encryption scheme incurs expensive in computational costs. In this solution, we employ two-server to obtain more efficient privacy-preserving TRACLUS protocol and lower the workload of the data owner in the 2PC-based solution with a single-server (See Chapter 7).

3. PRIDA that is a privacy-preserving data aggregation solution combines multi-key FHE with 2PC and threshold FHE with 2PC. Thanks to the use of these two cryptographic building blocks with a setting involving two non-colluding cloud servers (we name them as Aggregators in PRIDA), PRIDA supports scenarios with more than one *data analyser* who is interested in receiving the aggregation result. Furthermore, PRIDA enables data owners to have some control over which data analyser can have access to the resulting aggregated information. Moreover, with the introduction of an anonymous counting phase, data analysers can discover the aggregation result only when a sufficient number of data owners (more than a pre-defined threshold) authorise them (See Chapter 8).

To sum up, we believe that our proposed solutions meet the need for privacy-preserving protocols for machine learning techniques, enable data privacy with the realistic threat model, and achieve a good balance between privacy, efficiency, and accuracy/quality evaluation.

1.5 Organisation

The remaining of this thesis is organised as follows:

In Chapter 2, we introduce machine learning techniques, namely: Neural network, (trajectory) clustering, and data aggregation that we study in our solutions.

In Chapter 3, we introduce the building blocks that the newly designed privacy-preserving protocols discussed in the next chapters make use of, namely: Secure multi-party computation, homomorphic encryption, homomorphic proxy re-encryption, multi-key homomorphic encryption, and threshold homomorphic encryption.

The reader then can move on to Part I of this thesis whereby we study and identify the challenges, review the state-of-the-art, and introduce our solutions for privacy-preserving neural network classification and trajectory clustering in the scenario involving a single cloud server in Chapters 4 and 5. In Part II, we investigate a scenario involving two non-colluding cloud servers to reduce the computational and communication burden of the queriers/data owners when performing the privacy-preserving variants of neural network classification, trajectory clustering, and data aggregation in Chapters 6, 7, and 8.

Finally in Chapter 9, we conclude with the results of this dissertation and we discuss future research avenues.

Chapter 2

Machine Learning Techniques

Artificial Intelligence, deep learning, machine learning —whatever you're doing if you don't understand it— learn it. Because otherwise you're going to be a dinosaur within 3 years.

Mark Cuban

In this chapter, we introduce and study three machine learning techniques: Neural network, (trajectory) clustering, and data aggregation.

Machine learning can be defined as designing a system that is able to learn and solve problems based on the knowledge from its environment. Recent advances in Information Technology enable several companies to become more and more data-driven and collect/share/process more and more information about their clients. These data-driven companies employ *machine learning techniques* to process these data, which makes machine learning increasingly valuable. Machine learning techniques help these companies easily utilise their abundant data originating from their clients to design a system that is capable of solving problems, learning from the collected data, and so making (business related) decisions (or predictions) such as speech recognition [10], forecasting [11], or image classification [12], and improving their (customer) services.

The literature divides machine learning techniques into two subfields regarding the nature of the underlying technique's result/output: (i) *supervised machine learning techniques*, which build a model on top of a set of data having labels; and (ii) *unsupervised machine learning techniques*, which group unlabelled data into small groups regarding their similarities.

Supervised machine learning (*a.k.a* supervised learning) can be defined as a mapping from input data to some output data and further the computation of a mathematical model based on this resulting map called building a learning model, i.e., creating the

model from matching the input data indicated some label with the expected output label until the model is fitted appropriately [4]. In order to make this matching, a dataset for the supervised learning technique can be divided into three sub-datasets: (i) the training dataset, which is used to create/train the machine learning model, i.e., the resulting model learns from this dataset to optimise itself by checking the match of the input label and the output label of data; (ii) the cross-validation dataset, which is utilised to avoid/eliminate error in data that is not classified correctly by the currently trained model; and (iii) the test dataset employed for checking how much accurate the trained model is before its use for the real-life use cases. The datasets are usually split in the ratio of $60 : 20 : 20$, or even sometimes the dataset is split into two sub-datasets, namely training and test datasets in the ratio of $80 : 20$. Neural networks [6], support vector machines [13], random forest [14], linear regression [15], logistic regression [16], etc., are the examples of supervised learning techniques.

In an unsupervised machine learning technique (*a.k.a* unsupervised learning), the data do not have any label showing that the underlying data belongs to some subset having the same (or similar) properties in the supervised machine learning techniques such as neural networks. The data in the neural network are used to train a neural network model which can further be employed to classify new data. In unsupervised machine learning technique, one does not train any model (or learn from data) but aims to detect the hidden pattern(s), similarities, or differences among unlabelled data. Clustering is one of the unsupervised learning techniques that we study in this thesis.

Moreover, we also investigate another simple but useful statistical technique in the scope of this thesis, namely (iii) *data aggregation*, which gathers data from multiple sources and enables one to perform some statistics (sum, average, etc.) over them. Note that data aggregation is usually utilised as an initial step for machine learning techniques.

2.1 Neural Networks

This section focuses on the neural network as a supervised learning method; hence, we define it and describe its underlying functions (or operations).

A neural network [6] is a layered machine learning technique consisting of interconnected processing units called *neurons* that compute specific functions in each layer. The first layer of the neural network is defined as the input layer and the last layer is the output layer. The layers in between are named hidden layers. Each hidden layer evaluates a function over the output of the previous layer and obtains a result which becomes the input to the next layer. These hidden layers usually consist of either linear operations such as matrix multiplications and additions (fully connected layers or convolutional layers) or more complex operations such as `Sigmoid` [17] or `Max` [18] computation (activation or pooling layers).

More formally, a hidden layer can be defined as follows: It takes an input $\mathbf{X}$, evaluates a function f on the input along with a weight matrix $\mathbf{W}$ (and a bias vector $\mathbf{B}$), and outputs $\mathbf{Y}$ to the subsequent layer. Figure 2.1 illustrates an overview of the neural network structure.

Neural networks are usually designed and used in two phases: the training phase and

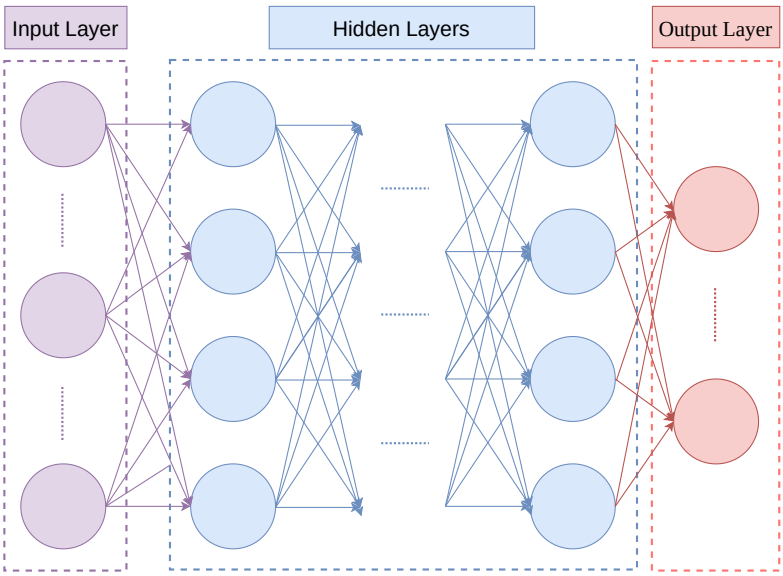


Figure 2.1 – An overview of the neural network structure

the classification (*a.k.a.* prediction) phase. The training phase is the learning process of using many data (and also their labels) from the training dataset, developing the neural network model which can learn from these data without any human intervention, and deriving and optimising the model parameters (e.g., weight matrix $\mathbf{W}$, bias vector $\mathbf{B}$, the number of hidden layers, each of them containing the number of neurons, etc.). The increase on the number of data items in the training dataset or the number of iterations (called epochs) make neural networks learn more and thus become more accurate; yet, note that having more epochs may increase the complexity of the training phase. The classification phase consists of performing predictions/classifications on future data items using the trained model.

The research on neural networks dates back to 1980s [19], yet they had not been commonly used due to their long training times. Thanks to the recent technological advances and the adaptation of GPUs in computation systems, the training time for neural networks is reduced significantly [20]. The improvement in performance triggered the popularity of neural networks, which in turn provided an outstanding success in some fields such as image classification [20, 21], face recognition [22], and board games [4].

Below, we describe the most common hidden layers used in neural networks:

2.1.1 Convolutional Layer

The Convolutional Layer (`Conv`, optional layer) aims to slide a filter, or a kernel, over the original input in order to obtain information about the similarity between the chunk of the original input covered by the filter and the filter itself. Figure 2.2 shows the operation of a convolutional filter on input $\mathbf{X}$.

The `Conv` layer applies filter $\mathbf{W}$ (or called as the weight matrix) to submatrix $\mathbf{X}'$ of

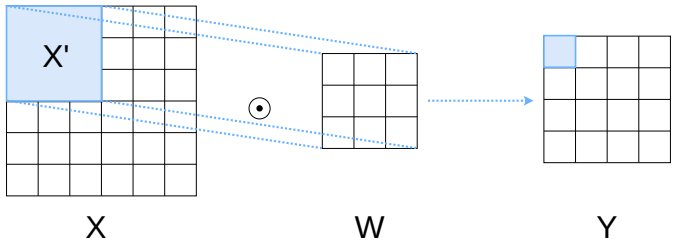


Figure 2.2 – Convolutional filtering in convolutional layer

the original input $\mathbf{X}$. $\mathbf{W}$ is slided every time to work on each index of the input matrix. The size of the input can be adjusted to fit to the filter size by appending some border values which is called padding. The operation performed in the `Conv` layer is a matrix multiplication ($\odot$) as follows:

$$\begin{aligned} \texttt{Conv}(\mathbf{X}, \mathbf{W}) &= \mathbf{X}' \odot \mathbf{W} \\ &= \sum x'_{i,j} \cdot w_{i,j}. \end{aligned} \tag{2.1}$$

2.1.2 Fully Connected Layer

The Fully Connected Layer (`FC`) connects each neuron in the current layer to each neuron in the previous layer along with a weight value. The underlying operations can be defined as a matrix multiplication and a vector addition as depicted in Equation 2.2:

$$\begin{aligned} \texttt{FC}(\mathbf{X}^t, (\mathbf{W}, \mathbf{B})) &= \mathbf{X}^t \odot \mathbf{W} + \mathbf{B}, \\ &= \mathbf{X}^{t+1} \end{aligned} \tag{2.2}$$

where $\mathbf{W}$ is the weight matrix, and $\mathbf{B}$ is the bias vector, $\mathbf{X}^t$ is the input of the current layer t, and $\mathbf{X}^{t+1}$ is the input of the next layer $t+1$.

2.1.3 Activation Layer

The Activation Layer (`Act`) is a nonlinear function. The goal of the activation layer is to check whether the pattern presents at a given position in the input data. In this layer, a nonlinear activation function ($\texttt{Act}(x_{i,j}) = y_{i,j}$ where $y_{i,j}$ is a neuron of the next layer) is applied to each neuron (i.e. $x_{i,j}$) of the input layer.

There exist different activation functions. We enumerate the commonly used ones in Equations 2.3, 2.4, and 2.5:

$$\textit{Sigmoid: } y_{i,j} = \frac{1}{1 + e^{-x_{i,j}}}, \tag{2.3}$$

$$\textit{Hyperbolic tangent (tanh): } y_{i,j} = \frac{e^{2x_{i,j}} - 1}{e^{2x_{i,j}} + 1}, \tag{2.4}$$

$$\textit{Rectified Linear Units (ReLU):}\ y_{i,j} = \max(0, x_{i,j}). \tag{2.5}$$

Note that the ReLU function is currently the mostly used activation function [5]. There are also some variants [6], such as the parametric version (PReLU) and the Exponential Linear Units (ELU).

2.1.4 Pooling Layer

The Pooling Layer (Pool) is a scaling layer which reduces the size of the input to make the neural network more optimised (decreases the size of the previous layer's output, usually the Conv layer). Reduction is performed by sliding a filter on the input matrix and performing the pooling operation on each area that is covered by the filter. Note that unlike the Conv and FC layers, operations of the pooling layer are nonlinear.

The two common types of pooling are:

(i) Max pooling where the maximum value within the area covered by the filter is selected; and

(ii) Average pooling where the average of the values within the area covered by the filter is selected.

2.1.5 Complementary Functions in Neural Networks

This section defines the complementary neural networks functions employed during training and/or classification phases.

Cost function

If once the learning process is completed, the error rate calculating from the matches of the label of inputs and its corresponding outputs' label is too high, the trained neural network model needs to be revisited. For this aim, a cost (or loss) function is employed to reduce this error rate and is periodically calculated during the training phase. An example of a cost function is the *Squared error loss* L defined in Equation 2.6.

$$\textit{Squared error loss:}\ L(y_{i,j}, x_{i,j}) = (y_{i,j} - x_{i,j})^2. \tag{2.6}$$

Stochastic Gradient Descent

Backpropagation is a process of adjusting the neural network parameters, namely weight matrices and bias vectors, in the existence of a non-negligible error rate during the learning phase. This method simply takes the derivative of the defined cost function for these underlying parameters and adjusts them accordingly using *Stochastic Gradient Descent* (SGD) [5], which is an iterative calculation to optimise these neural network parameters.

Batch Normalisation

A batch normalisation (BN) [5] function applies *batch normalisation* usually before the activation layer to normalise the neurons in the current layer to the next layer, to make the NN model more stable while learning, and to reduce the number of epochs (i.e., each iteration of the current NN structure and its learnt parameters from the previous iterations) during training. A batch normalisation layer can be defined in Algorithm 1 as follows:

Algorithm 1 Batch Normalisation

Input: Neurons $x_{i,j}$ over a mini-batch $B = \{x_{1,j}, \ldots, x_{i,j}, \ldots, x_{m,j}\}$ and batch normalisation parameters γ and β

Output: $y_{i,j} = BN_{\gamma,\beta}(x_{i,j})$

$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_{i,j}$ // mini-batch mean

$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_{i,j} - \mu_B)^2$ // mini-batch variance

$\hat{x}_{i,j} \leftarrow \frac{x_{i,j}-\mu_B}{\sqrt{\sigma_B^2+\epsilon}}$ // normalise

$y_{i,j} \leftarrow \gamma\hat{x}_{i,j} + \beta$ // scale and shift

Softmax and Argmax

Neural networks employ some functions such as *Softmax* in Equation 2.7 or *Argmax* in Equation 2.8. These functions are implemented as a neural network layer, just after the output layer (but sometimes they can be considered as the output layer). Note that the output layer should contain the same number of neurons as the number of labels in the dataset. While Softmax computes a probability ($Pr(.)$) for every possible class of the given dataset, Argmax results in 1 for the index of the largest neuron value in the output layer; otherwise, 0. The Argmax function is usually used as an alternative to Softmax.

$$\textit{Softmax: } Pr(x_{i,j}) = \frac{e^{x_{i,j}}}{\sum e^{x_{i,j}}}, \tag{2.7}$$

$$\textit{Argmax: } \texttt{Argmax}(x_{0,j}, \ldots, x_{i,j}, \ldots) = (0, \ldots, 0, 1, 0, \ldots). \tag{2.8}$$

2.1.6 Neural Network Model Structure

The previously defined layers are assembled in the neural network model denoted by **M** and previously depicted in Figure 2.1 as follows: (i) The neural network model architecture can start with some number of convolutional layers or fully connected layers, and each of them is followed by some nonlinear layer (an activation layer and/or a pooling layer); (ii) This sequence of these underlying layers can be applied some number of repetitions; and (iii) Once these are executed, the fully connected layer is usually applied as the output layer and followed by a `Softmax` (or an `Argmax`) layer.

A neural network model **M** can be described as a composite function since the layers of the model are created by composing one layer into another one: The outer function

can be considered as the output layer operations, and the first function in the side of the other functions will be the input layer operations. In other words, a neural network seems a function taking weight matrices and bias vectors as inputs and a data $\mathbf{X}$ to classify, and outputs the probability of each label for this data. For example, if one holds a model $\mathbf{M}$ consisting of 1 `FC` layer, 1 `ReLU` the activation layer followed by 1 `FC` layer and 1 Softmax function as an output layer. This model can be formulated as follows:

$$\mathbf{M}(\mathbf{X}) = \texttt{Softmax}(\texttt{FC}(\texttt{ReLU}(\texttt{FC}(\mathbf{X})))). \tag{2.9}$$

If a neural network contains some convolutional (`Conv`) layer(s), then it is named Convolutional Neural Network (CNN) as shown in Equation 2.10. Note that CNN usually starts with a `Conv` layer and is employed for classifying images. The CNN architecture can be defined as follows:

$$\mathbf{M}' = [[\texttt{Conv} \rightarrow \texttt{Act}]^p \rightarrow \texttt{Pool}]^q \rightarrow [\texttt{FC}]^r, \tag{2.10}$$

where p, q, and r are some integers. Note that the model notation in Equation 2.10 can be used for the previous model $\mathbf{M}$ (formulated in Equation 2.9), which does not contain any `Conv` layers: $\mathbf{M} = \texttt{FC} \rightarrow \texttt{ReLU} \rightarrow \texttt{FC} \rightarrow \texttt{Softmax}$.

2.1.7 Neural Networks Accuracy Evaluation

The neural network as a supervised learning technique is built over a large number of data with labels. In order to build an accurate neural network model as previously mentioned, the dataset is divided into different sub-datasets. Generally, while the training dataset is utilised when building the model, the test dataset defines how much accurate the created model is. In order to come up with a model that accurately predicts the actual input label, one first trains some models and evaluates the performance of the built models on the training dataset. The evaluation results are used over the cross-validation dataset to compare the performance between different NN models and choose the most efficient one. Finally, performance tests of the chosen model are performed on the test dataset. One of the performance evaluations of the underlying model is *prediction accuracy*, and it can be defined as the determination of which class a given input belongs to and be computed by multiplying 100 with the number of correct predictions divided by the total number of predictions.

This PhD thesis employs the most frequently used method to calculate the accuracy of our neural network models, namely a Confusion matrix.

Confusion Matrix

A *confusion matrix* (CM) summarises the prediction results of the neural network model: This accuracy calculation method gives better understanding about the trained model if making any error, and the overall accuracy of the underlying model is computed using the following formula in Equation 2.11:

$$\sum_{x \in Classes} freq(x) \times P(predict = y/class = x). \tag{2.11}$$

where $freq(x)$ represents the frequency of the class x in the test dataset. For each $y \in Classes$ and $x \in Classes$, the confusion matrix presents the probability that the model predicts the class y knowing that the input belongs to class x (i.e., $P(predict = y/class = x)$).

2.2 Clustering

Clustering is an unsupervised machine learning technique that is a procedure of grouping a set of elements into classes of *similar* elements. Clustering can be used to segment customers of some company into alike demographics, or detect anomalies, or make very large datasets simple by categorising them into small subgroups regarding their related features. k-means [23], DBSCAN [24], and TRACLUS [7] are some clustering methods: While the k-means algorithm is based on centroids which are computed by taking the mean of all the elements in the underlying cluster, DBSCAN and TRACLUS rely on the density of the dataset elements: clusters are arbitrarily shaped, and each of their elements is connected to each other.

In this section, we introduce these three data clustering algorithms: namely, k-means, DBSCAN, and finally TRACLUS; yet, it is worth to note that our main focus is TRACLUS for this PhD thesis.

2.2.1 k-means

This algorithm is proposed by Macqueen et al. [23] and is one of the unsupervised machine learning techniques. k-means iteratively divides the dataset into k subsets. As the pseudocode depicts in Algorithm 2, it starts with randomly creating (or chosen from the dataset) k cluster centroids $c_1, c_2, \ldots, c_k$ which represent the centre of the k clusters. Then, each data element in the dataset is assigned to the one among k clusters that have a minimum distance between the centroid and itself. In general, the distance computation is performed using the Euclidean distance; however, other distance measurements can be used [25]. After all elements are assigned in the dataset, the k centroids are updated by computing the arithmetic mean of the elements assigned to their clusters. The next iteration again assigns the elements to the closest cluster and then updates the centroids of each cluster. The algorithm stops if the centroids do not change anymore, or they may change little, or a predetermined number of iterations is reached.

The complexity of k-means is calculated by $\mathcal{O}(kndI)$ where k is the number of clusters, n is the number of points in dataset D, d is the dimension of the points, and I the number of iterations that the algorithm ran.

Algorithm 2 k-means algorithm

Input: Dataset D and the number of clusters k

Output: Set of clusters $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ and set of corresponding centroids $\mu = \{\mu_1, \ldots, \mu_k\}$

Initialisation: Select k initial centroids

repeat

for *each data point* $i \in D$ **do**

  Find the closest centroid μ_j // Euclidean distance computation

  Assign i to cluster c_j

for *each cluster* c_j **do**

  Update the centroid μ_j // mean computation

until stop criterion // Maximum number of iterations is reached or centroids are not significantly changed.

2.2.2 DBSCAN

Density Based Spatial Clustering of Applications with Noise (DBSCAN) proposed by Ester et al. [24] is another clustering technique that identifies the class in datasets and cannot only discover the arbitrarily shaped cluster but also identify the noise points.

This technique/algorithm takes two inputs: $MinPts$ that indicates the minimum number of points in a cluster, and threshold ϵ, indicating the maximum distance between each point in a cluster. DBSCAN depicted in Algorithm 3 initiates with a random point in a dataset of points D and then investigates its neighbourhood with a given threshold value of ϵ, i.e., the neighbourhood around a point i can be defined as a sphere of radius ϵ and its center, $i : N_\epsilon(i) = \{j \in D | dist(i,j) \leq \epsilon\}$, where $dist(.,.)$ is the Euclidean Distance. If the neighbourhood around this point satisfies the minimum number of points ($|N_\epsilon(i)| \geq MinPts$), i becomes a core point. Later, this neighbourhood with this core point also becomes a cluster c, and further, the algorithm searches for other points until all points around its neighbourhood join this cluster. There can be some points that are not grouped: The algorithm selects ungrouped points and proceeds again for them. Once all the searches finish, some points neither core points nor the neighbourhood of any core point are labelled as *noise*.

If n points in dataset D, the complexity of the algorithm is $\mathcal{O}(n^2)$. As previously mentioned, DBSCAN is not the main focus of this thesis. It is presented in order to easily introduce TRACLUS, a trajectory clustering technique that is based on DBSCAN.

2.2.3 TRACLUS

TRAjectory CLUStering (TRACLUS) is a density-based clustering algorithm particularly optimised for clustering trajectories[1] proposed by Lee et al. [7]. The idea behind TRACLUS is based on the previously introduced DBSCAN, which segments the trajectories into the line segments and then uses similar techniques as in DBSCAN to cluster these line segments. TRACLUS uses line segments instead of points in DBSCAN. Therefore,

[1] Sequences of multidimensional points.

Algorithm 3 DBSCAN algorithm

Input: Dataset D, the threshold ϵ, and $MinPts$
Output: Set of clusters $\mathcal{C} = \{c_1, c_2, \ldots, c_l\}$
repeat
Initialisation: Select a point i
if $|N_\epsilon(i)| \geq MinPts$ **then**
Label i as a core point of cluster c_1
Check point j in $N_\epsilon(i)$
Add j in c_i
Label j either as a core point or a border point
until No more points added to c_i
if *A point j is not either a core or border point* **then**
Label j as a noise

$MinPts$ is renamed as $MinLns$ in TRACLUS and the functionality of ϵ remains the same as in DBSCAN.

TRACLUS consists of two phases: *Partitioning* and *Grouping (*a.k.a *Clustering).* In the first phase, called *partitioning* phase, the algorithm divides trajectories into sub-trajectories that are called *line segments.* Each line segment is represented by two points. This partitioning phase should ensure that the output is close to the original trajectory (*preciseness*) and that the number of line segments is as small as possible (*conciseness*). The more line segments are created, the better line segments approximate the original trajectory. However, preciseness and conciseness will contradict in this case. Therefore, the partitioning part of TRACLUS aims at finding the best trade-off between preciseness and conciseness.

The characteristic points, marking the course of the trajectory changing significantly, play an important role to approximate the trajectory and thus build the best trade-off between preciseness and conciseness. If characteristic points were used at each point of the trajectory, this would make the line segments perfect; however, this further would contradict with the conciseness that means using as few line segments as possible. The characteristic points are calculated with the Minimum Description Length (MDL) principle [26]. MDL consists of two components: $L(H)$ and $L(D|H)$: (i) $L(H)$ in Equation 2.12 sums up all the lengths of line segments and measures the conciseness; and (ii) $L(D|H)$ in Equation 2.13 analyses the difference between the original trajectory and its partitions, and measures the preciseness.

$$L(H) = \sum_{j=1}^{par_i - 1} log_2(len(p_{c_j} p_{c_{j+1}})) \tag{2.12}$$

where $len(p_{c_j} p_{c_{j+1}})$ describes the Euclidean distance between p_{c_j} and $p_{c_{j+1}}$.

$$L(D|H) = \sum_{j=1}^{par_i-1} \sum_{k=c_j}^{c_{j+1}-1} \{log_2(d_\perp(p_{c_j}p_{c_{j+1}}, p_{k}p_{k+1})) + log_2(d_\theta(p_{c_j}p_{c_{j+1}}, p_{k}p_{k+1}))\} \quad (2.13)$$

where D is the dataset, and $d_\perp$ and d_θ are parts of the distance metric that are explained in the clustering phase.

To find the optimal trade-off between preciseness and conciseness, it is necessary to evaluate the MDL cost for each possible subset of the characteristic points in a trajectory. Since the partitioning algorithm aims at finding a local minimum for the MDL cost, the algorithm compares the costs for creating a partition $\text{MDL}_{par}(p_i, p_j)$ with for not creating a partition $\text{MDL}_{nopar}(p_i, p_j)$ between two points p_i and p_j of a trajectory. If $\text{MDL}_{par}(p_i, p_j)$ is smaller than $\text{MDL}_{nopar}(p_i, p_j)$, this means that the cost of adding a characteristic point at p_j is less than the cost of not adding a characteristic point. The partitioning phase in Algorithm 4 iterates through a trajectory and checks each point of a trajectory from the initial point of it whether the partitioning becomes more expensive than the non-partitioning. If the cost is not high, the algorithm takes the next point of the trajectory and analyses its cost with respect to the initial point. Otherwise, the algorithm adds a new characteristic point before the current ending point since in this point non-partitioning is still more costly than partitioning. This newly added characteristic point becomes the initial point of the new line segment.

Algorithm 4 Partitioning a trajectory

Input: Dataset D containing trajectories $TR_i = p_1p_2 \dots p_j \dots p_{len_1}$
Output: Set of characteristic points $CP_1 = \{cp_1, cp_2, \dots, cp_l\}$

```
Add p_1 into the set CP_1
start_index = 1, length = 1
while start_index + length ≤ len_i do
    curr_index = start_index + length
    cost_par = MDL_par(p_start_index, p_curr_index)
    cost_nopar = MDL_nopar(p_start_index, p_curr_index)
    if cost_par > cost_nopar then
        Add p_curr_index into the set CP_i
        start_index = curr_index − 1, length = 1
    else
        length = length + 1
Add p_len1 into the set CP_1
```

Once the partitioning phase is done, line segments are clustered with a density-based approach during *the Clustering phase*. TRACLUS uses DBSCAN (See Section 2.2.2), and therefore, it also requires two parameters $MinLns$ and ϵ. The optimum way for the selection of these parameters is the heuristic defined in [7]. This heuristic consists of finding out the optimal epsilon value using the simulated annealing algorithm [27]: ϵ is set as the value that minimises the entropy of the clustering [7]. Note that *entropy* investigates the consistency in discrimination between the line segments.

TRACLUS uses different distance metrics composed of a tripartite distance metric and defined in Equations 2.14, 2.15, and 2.16 to measure the distance between line segments (represented by two points, namely starting point s_i and ending point e_i) in Figure 2.3 instead of the Euclidean distance in DBSCAN [24]. Similarly to DBSCAN, TRACLUS effectively identifies not only the arbitrary shape of clusters but also the noise line segments.

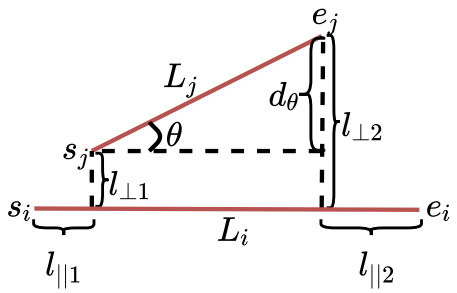


Figure 2.3 – Tripartite distance metric in TRACLUS

Equation 2.14 is the perpendicular distance based on Lehmer mean defined by $L_p(x_1, x_2, ..., x_n) = \frac{\sum_{k=1}^{n} x_k^p}{\sum_{k=1}^{n} x_k^{p-1}}$ that measures the vertical distances $l_{\perp 1}$ and $l_{\perp 2}$ between two line segments, namely L_i and L_j.

$$d_\perp = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}} \tag{2.14}$$

The parallel distance in Equation 2.15 is the horizontal distances $l_{\|1}$ and $l_{\|2}$ between two line segments L_i and L_j.

$$d_\| = min(l_{\|1}, l_{\|2}) \tag{2.15}$$

The last distance metric in Equation 2.16 is the angular distance d_θ that measures the directional difference between two line segments. The angle θ is the smaller intersection angle between L_i and L_j.

$$d_\theta = \begin{cases} \|L_j\| \times \sin(\theta) & \text{if } 0 \leq \theta < 90, \\ \|L_j\| & \text{if } 90 \leq \theta \leq 180 \end{cases} \tag{2.16}$$

TRACLUS uses this tripartite distance metric in Equation 2.17 inspired by the area of pattern recognition [5] to determine close line segments and to regroup them into one cluster.

The TRACLUS distance metric denoted by $dist(L_i, L_j)$ in Equation 2.17 is the result of a weighted average of these three distance metrics, and $w_\perp$, $w_\|$, and w_θ are the weights assigned to each distance, that are set to 1 by default.

$$dist(L_i, L_j) = w_\perp \cdot d_\perp(L_i, L_j) + w_\parallel \cdot d_\parallel(L_i, L_j) + w_\theta \cdot d_\theta(L_i, Lj) \tag{2.17}$$

TRACLUS provides some definitions for the neighbourhood notion to define a cluster:

Definition 2.2.3.1 The *ϵ-neighborhood* $N_\epsilon(L_i)$ of a line segment L_i in dataset D is defined by $N_\epsilon(L_i) = \{L_j \in D | dist(L_i, L_j) \leq \epsilon\}$.

Definition 2.2.3.2 A line segment $L_i \in D$ is called a *core line segment* with respect to ϵ and $MinLns$ if $|N_\epsilon(L_i)| \geq MinLns$.

Definition 2.2.3.3 A line segment $L_i \in D$ is *directly density-reachable* from a line segment $L_j \in D$ with respect to ϵ and $MinLns$ if (1) $L_i \in N_\epsilon(L_j)$ and (2) $|N_\epsilon(L_j)| \geq MinLns$.

Definition 2.2.3.4 A line segment $L_i \in D$ is *density-reachable* from a line segment $L_j \in D$ with respect to ϵ and $MinLns$ if there is a chain of line segments $L_j, L_{j-1}, ..., L_{i+1}, L_i \in D$ such that L_k is directly density-reachable from L_{k+1} with respect to ϵ and $MinLns$.

Definition 2.2.3.5 A line segment $L_i \in D$ is *density-connected* to a line segment $L_j \in D$ with respect to ϵ and $MinLns$ if there is a line segment $L_k \in D$ such that both L_i and L_j are density-reachable from L_k with respect to ϵ and $MinLns$.

Definition 2.2.3.6 A non-empty subset $C \subseteq D$ is called a *density-connected set* with respect to ϵ and $MinLns$ if C satisfies the following two conditions:

1. *Connectivity:* $\forall L_i, L_j \in C$, L_i is density-connected to L_j with respect to ϵ and $MinLns$.
2. *Maximality:* $\forall L_i, L_j \in D$, if $L_i \in C$ and L_j is density-reachable from L_i with respect to ϵ and $MinLns$, then $L_j \in C$.

Definition 2.2.3.7 The set of *participating trajectories* of a cluster C_i is defined by $PTR(C_i) = \{TR(L_j) \mid \forall\ L_j \in C_i\}$. Here, $TR(L_j)$ denotes the trajectory from which L_j has been extracted. Then, $|PTR(C_i)|$ is called the *trajectory cardinality* of the cluster C_i.

In this phase, the clustering algorithm in Algorithm 5 takes each unclassified line segment and computes the distance from all other line segments that are in its neighbourhood after comparing their distance with ϵ. If this neighbourhood size is sufficiently large (i.e., larger than or equal to $MinLns$), it is grouped as a cluster. Then, the algorithm iterates through all neighbours that were already assigned to the new cluster. The algorithm further calculates their neighbourhood to determine whether they are *core line segments*. If so, they are added to the cluster and their neighbourhood will be checked. After finishing the iteration through all line segments, the algorithm checks whether the cardinality of the cluster is sufficient.

Whereas the complexity of the TRACLUS partitioning phase takes $\mathcal{O}(n)$ where n is the number of points of one trajectory, the time complexity of the clustering phase is $\mathcal{O}(n^2)$ where n is the number of line segments in D.

Algorithm 5 Clustering phase

Input : (1) A set of line segments $D = \{L_i, ..., L_{num_{ln}}\}$
(2) Two parameters ϵ and $MinLns$

Output : A set of clusters $\mathcal{C} = \{c_1, ..., c_{num_{clus}}\}$

Set $clusterId$ to be 0
Mark all the line segments in $\mathcal{D}$ as $unclassified$ **foreach** $L \in \mathcal{D}$ **do**
  if L *is* $unclassified$ **then**
    Compute $N_\epsilon(L)$**if** $|N_\epsilon(L)| \geq MinLns$ **then**
      Assign $clusterId$ to $\forall X \in N_\epsilon(L)$;
      Insert $N_\epsilon(L) - \{L\}$ into the queue $\mathcal{Q}$
      Expand Cluster($\mathcal{Q}$, $clusterId$, ϵ, $MinLns$)
      $clusterID+ = 1$;
    else
      Mark L as $noise$;
    end
  end
end
Allocate $\forall L \in \mathcal{D}$ to its cluster $c_{clusterId}$**foreach** $c \in \mathcal{C}$ **do**
  if $|PTR(c)| < MinLns$ **then**
    Remove c from the set of $\mathcal{C}$ of clusters
  end
end
ExpandCluster (*($\mathcal{Q}$, $clusterId$, ϵ, $MinLns$)*)
  while $\mathcal{Q} \neq \emptyset$ **do**
    Let M be the first line segment in $\mathcal{Q}$
    Compute $N_\epsilon(M)$**if** $|N_\epsilon(M)| \geq MinLns$ **then**
      foreach $x \in N_\epsilon(M)$ **do**
        if x *is* $unclassified$ *or* $noise$ **then**
          Assign $clusterId$ to x
        end
        if x *is* $unclassified$ **then**
          Insert x into the queue $\mathcal{Q}$
        end
      end
      Remove M from the queue $\mathcal{Q}$
    end
  end
end

2.2.4 Clustering Quality Evaluation

To evaluate the quality of a clustering method, several quality assessment measures have been proposed [28–30]. The commonly used clustering quality indices are the Silhouette Coefficient, the Silhouette Coefficient with Noise, and the Density-Based Clustering Validation which are defined in this section.

Silhouette Coefficient

As clustering is an unsupervised machine learning technique, a ground truth is usually not available. The most frequently used methodology to evaluate the quality of the clustering result is the silhouette analysis [28]. This methodology analyses the resulting clusters by evaluating the similarity between the elements within the cluster and the elements of other clusters. The analytical output is called the *Silhouette Coefficient* (SC) which is close to 1 for a good clustering.

The silhouette coefficient of each data item i in a resulting cluster c_j of the clustering technique, $1 \leq j \leq l$, is calculated as follows:

$$SC(i) = \frac{b(i) - a(i)}{max(a(i), b(i))}. \tag{2.18}$$

where $a(i)$ is the mean distance between element i and each of other elements within the same cluster, and $b(i)$ is the minimum mean distance between i and any of other elements in different clusters.

Silhouette Coefficient with Noise

Some elements may remain unclustered after the execution of the clustering algorithm. Such elements are considered as *noise*. Although SC allows powerful insights about how tightly elements are clustered, it does not take outliers that remain noise into account. Thus, an additional penalty measure is needed to evaluate the SC with noise as described in [29], namely SC_{noise}.

The SC with noise penalty is calculated as in Equation 2.19:

$$SC_{noise}(i) = SC(i).\left(\frac{p-n}{p}\right). \tag{2.19}$$

where p, i, and n are dataset size, each data item in a resulting cluster c_j, and the number of noisy elements, respectively.

Density-Based Clustering Validation

Another method specifically designed to assess the quality of density-based clustering algorithms is the Density-Based Clustering Validation (DBCV) [29,30]. In contrast to the standard SC, DBCV also takes noise into account. Similar to the other evaluation metrics, a higher DBCV indicates a good clustering quality.

The DBCV score of the clustering algorithm is calculated as follows:

$$DBCV(\mathcal{C}) = \sum_{i=1}^{l} \frac{|c_i|}{|\mathcal{P}|} \frac{\min\limits_{1 \leq j \leq l, j \neq i} (DSPC(c_i, c_j)) - DSC(c_i)}{max\left(\min\limits_{1 \leq j \leq l, j \neq i} (DSPC(c_i, c_j)), DSC(c_i)\right)}. \tag{2.20}$$

where $\mathcal{C} = \{c_1, c_2, \ldots, c_l\}$, $\mathcal{P} = \{L_1, L_2, \ldots, L_n\}$, DSPC and DSC correspond to the resulting clusters, the data items, the density separation for a pair of clusters, and density sparseness of a cluster, respectively.

2.3 Data Aggregation

Data aggregation can be defined as a data analytics process of collecting data from multiple sources (clients/users of some data-driven companies) and performing some statistical analysis such as sum, average, etc., over these collected data. Data-driven companies are always in competition to collect more and more information about their clients (or users) and further utilise them to improve their services. However, companies need the data to be gathered/expressed in a summary form. Thanks to data aggregation, these companies (or a third party cloud server) assemble data and execute some statistical analysis such as sum, average, etc.. Thus, this increasingly makes such aggregate data essential and valuable for companies since the aggregate data is ready to bring answers to analytical questions for groups of people or helps companies achieve insights about their business analysis. For example, consider a scenario whereby some sports companies would like to obtain some statistics (i.e., sum, average, etc.) over many customers' consumption. Customers can participate to these various statistics with their data such as the purchase history of sports from some online marketing website or the sport-related data from their smart device, a running app or a smartwatch: their location information from the marketing website or location history from the app shows that fitness-conscious clients go past near these sports companies when commuting between their home and office. These kind of information would be collected by some cloud server(s) who can aggregate, perform the required statistics over them, and send the aggregate result to corresponding sports companies. Therefore, the underlying companies with this valuable information would publish an advertisement to their target group and become more attractive to them.

2.4 Summary of Machine Learning techniques

In this chapter, three particular machine learning techniques have been presented, namely neural network, TRACLUS, and data aggregation. A neural network is composed of input, hidden, and output layers, and these layers consist of many neurons, which are simply some functions. These functions can contain some non-linear (or complex) operations such as sigmoid, division, etc. Moreover, TRACLUS also involves complex operations such as sine, division, etc. In the next chapters, we introduce some solutions to obtain

efficient and accurate/qualified evaluation results when designing privacy-preserving variants of these machine learning techniques based on cryptographic techniques.

Chapter 3

Cryptographic Techniques

Lots of people working in cryptography have no deep concern with real application issues. They are trying to discover things clever enough to write papers about.

Whitfield Diffie

In this chapter, we introduce the building blocks that the newly designed privacy-preserving protocols discussed in the next chapters make use of, namely: Secure multi-party computation, homomorphic encryption, homomorphic proxy re-encryption, multi-key homomorphic encryption, and threshold homomorphic encryption.

3.1 Security Notions

This section defines the security notions.

A *cryptosystem* (or an encryption scheme) converts cleartext (or called plaintext, needs to be protected) to ciphertext (encrypted plaintext) or ciphertext to plaintext to securely encode or decode these messages by employing the encryption or decryption algorithm with the key generation algorithm.

3.1.1 Notations

We write $x \in_R X$ to represent an element x being randomly sampled from some domain X, and $x \xleftarrow{\$} X$ to represent an element x being uniformly sampled from some distribution X.

3.1.2 Ideal/Real Simulation paradigm.

In the simulation paradigm, the ideal security setting is outsourcing inputs of both parties who run the protocol to a trusted third party who can perform the computations and return the output. In the real-world setting, the security goal is to show that if an adversary $\mathcal{A}$ can attack the protocol in the real world, then the attack can be also performed by an adversary $\mathcal{S}$ in the ideal world. Since the attacks of $\mathcal{S}$ are not successful in the ideal setting, the attacks in the real world also fail and the protocol is proved to be secure in the real world. In Definitions 3.1.1 and 3.1.2, we provide the formal definitions for security and indistinguishability from [31].

Definition 3.1.1 (Computational Indistinguishability)**.** Let $X(a,\kappa)$ and $Y(a,\kappa)$ be two probability ensembles where $a \in \{0,1\}^*$ is the input of the parties and κ is the security parameter. $X(a,\kappa)$ and $Y(a,\kappa)$ are *computationally indistinguishable* (i.e. $X(a,\kappa) \stackrel{c}{\equiv} Y(a,\kappa)$) if there exists a negligible function $\mu(\kappa)$ for every nonuniform polynomial time algorithm D, and for every $a \in \{0,1\}^*$ such that

$$|\Pr[D(X(a,\kappa)) = 1] - \Pr[D(Y(a,\kappa)) = 1]| \leq \mu(\kappa). \tag{3.1}$$

Definition 3.1.2 (Definition of Security)**.** P_1 and P_2 are two parties who want to run a protocol π over their inputs x and y, respectively to compute a functionality $f(x,y)$ which outputs $f_1(x,y)$ and $f_2(x,y)$ for P_1 and P_2, respectively. In the execution of π, the view of parties are

$$\mathbf{view}_1^\pi(x,y,\kappa) = (x, r_1; m_1, m_2, \cdots, m_t), \tag{3.2}$$
$$\mathbf{view}_2^\pi(x,y,\kappa) = (y, r_2; m_1, m_2, \cdots, m_t), \tag{3.3}$$

where r_1, r_2 are the randomness of the parties, κ is the security parameter and m_i's are the intermediary messages received by each party. The output of π is $\mathbf{output}^\pi(x,y,\kappa) = (\mathbf{output}_1^\pi(x,y,\kappa), \mathbf{output}_2^\pi(x,y,\kappa))$, such that $\mathbf{output}_1^\pi(x,y,\kappa)$ and $\mathbf{output}_2^\pi(x,y,\kappa)$ are the local outputs of P_1 and P_2. We say that π securely computes $f(x,y)$ in the presence of semi-honest, non-adaptive, computationally bounded adversaries, if there exist probabilistic polynomial-time simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ such that

$$\{\mathcal{S}_1(1^\kappa, x, f_1(x,y)), f(x,y)\} \stackrel{c}{\equiv} \{\mathbf{view}_1^\pi(x,y,\kappa), \mathbf{output}^\pi(x,y,\kappa)\}, \tag{3.4}$$
$$\{\mathcal{S}_2(1^\kappa, y, f_2(x,y)), f(x,y)\} \stackrel{c}{\equiv} \{\mathbf{view}_2^\pi(x,y,\kappa), \mathbf{output}^\pi(x,y,\kappa)\}. \tag{3.5}$$

3.1.3 Adversarial Models and Attacks

The adversarial models for a security protocol can be categorised into three groups [32]:

Semi-honest adversarial model. In this model, all parties follow the protocol steps correctly but they can try to extract any information during the execution of the protocol. This security model guarantees that no data is leaked. A semi-honest adversary is also named honest-but-curious or passive.

Malicious adversarial model. In this adversarial model, parties can act maliciously to deviate from the protocol steps. This model guarantees that no adversarial attack can be successful. A malicious adversary is also called active.

Adaptive adversarial model. An adversary arbitrarily chooses to corrupt some honest party (act maliciously) at any time during the protocol execution. Once the party is corrupted, it is assumed corrupted from that point on.

3.1.4 Chosen Plaintext Attack

An attacker can obtain ciphertexts for arbitrary plaintext information. If the attacker cannot guess whether the given ciphertext is an encrypted plaintext m_0 or m_1 with a probability more than 1/2, then this encryption scheme is called Chosen Plaintext Attack (CPA)-secure. Moreover, the encryption scheme can be defined as indistinguishable, and the encryption does not disclose any information regarding plaintexts. Even when the same plaintext is encrypted two times, the encryption scheme outputs two different ciphertexts; therefore, it is called semantically secure (or IND-CPA secure) [33].

3.2 Secure Multi-party Computation

Secure multi-party computation is introduced in early 1980s by Yao [34, 35] with the definition of secure two-party computation (2PC) and defines Yao's Millionnaire problem: Two millionnaires want to learn who is richest one without disclosing their actual wealth. They solve this problem by comparing their wealth using secure computation to ensure that they learn only the richest one and nothing else is revealed. Then, the problem was generalised to multiple parties by Goldreich et al. in [36].

Secure multi-party computation (MPC) is defined as a system in which a group of data owners can jointly compute a function of their private inputs without disclosing the underlying inputs, but the output of the function. Formally, let $P_1, \ldots, P_n$ be n parties and each of them having input $x_1, \ldots, x_n$, respectively. As illustrated in Figure 3.1, these parties want to jointly compute function f over all inputs $\{x_1, \ldots, x_n\}$ and learn the output without revealing their input.

MPC should ensure the following two properties [37], at least: (i) *input privacy*, i.e., parties' inputs should remain private and only the output of the function is learned; and (ii) *correctness*, i.e., even if some parties maliciously act, the correct output is obtained.

Existing MPCs can leverage Oblivious Transfer (OT) [38], Yao's Garbled circuits [34, 35], or secret sharing (additive or Boolean) [39] which are defined in the next section.

3.2.1 Oblivious Transfer

OT [38] is a fundamental cryptographic primitive that is used as a building block in MPC. OT allows a party to choose and retrieve k out of n secrets from another party without disclosing which secrets have been chosen. Usually, the 1-out-of-2 OT is used, ensuring that one secret out of two is retrieved: Let Alice have two inputs x_0 and x_1,

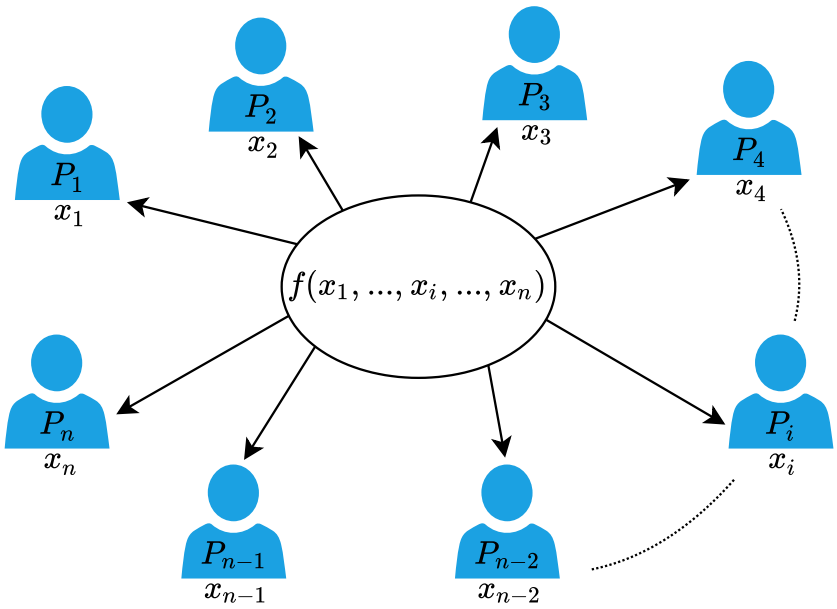


Figure 3.1 – Secure multiparty computation

and Bob select a bit b and want to obtain x_b. OT ensures that Bob does not learn x_{1-b} and Alice does not discover b.

3.2.2 Yao's Garbled Circuits

Yao's sharing (*a.k.a.* Garbled Circuits (GC)) is a secure two-party computation that allows two parties, Alice and Bob, each of them holding one input x_1 or x_2, to jointly evaluate a function $f(x_1, x_2)$ in the presence of a semi-honest adversary. Yao's GC protocol contains three steps: (i) Converting the function f into a Boolean circuit composed of addition and/or multiplication gates (i.e., a gate is a circuit that realises a Boolean operation such as AND); (ii) garbling the Boolean circuit; and (iii) evaluating the garbled circuit. Let Alice be the Garbler and Bob be the Evaluator. While Steps (i) and (ii) are performed by Alice, Bob computes the last step, namely Step (iii). In more details, Alice builds a garbled version of a circuit for the function f by obfuscating all possible outputs: Alice, the garbler, assigns two keys that correspond to bit values 0 and 1 for each wire of the circuit. Then, Alice computes four ciphertexts for each binary gate with the input and output wires. After obtaining ciphertexts, Alice randomly orders (or permutes) these four outcoming values. The resulting garbled circuit and Alice's garbled input $\mathrm{GI}(x_1)$ are sent to Bob. Alice also provides a map from the garbled-circuit outputs to the actual bit values. Once receiving this circuit, Bob, the evaluator, uses 1-out-of-2 OT [38] with Alice to obliviously obtain his garbled circuit value $\mathrm{GI}(x_2)$ without revealing it to Alice. Bob further evaluates the function $f(x_1, x_2)$ using $\mathrm{GI}(x_1)$ and $\mathrm{GI}(x_2)$.

3.2.3 Secret Sharing

Alternatively to Yao's Garbled Circuits, MPC solutions based on secret sharing consist of distributing secrets among parties involved in the system and further evaluating the function defined as a circuit accordingly.

Boolean secret sharing

The GMW protocol [36] relies on Boolean shares and mainly support XOR operations over single bits. The evaluated function is encoded as a Boolean circuit, and OT is used during the evaluation of the circuit. The Boolean circuit takes as inputs bit x from Alice and bit y from Bob. These bits are first secret-shared between the parties as $x = x_1 \oplus x_2$ and $y = y_1 \oplus y_2$, where shares $x_1, y_1 \in_R \{0,1\}$ are distributed to Alice and shares $x_2, y_2 \in_R \{0,1\}$ to Bob. Then, both parties evaluate the circuit gate by gate. For instance, given shared values, an XOR gate with input bits x and y and output bit z is evaluated locally (i.e. without communication) by each party by computing $z_i = x_i \oplus y_i$. Value z can be retrieved by exchanging and XORing the shares. The evaluation of AND gates is more challenging and requires Alice and Bob to interact.

When using Boolean shares, inputs are shared in $\mod 2$ for the operations of addition and multiplication. We denote it by $\mathrm{BS}.\mathrm{Share}(p, d)$ returning $\langle d \rangle_1, \ldots, \langle d \rangle_p$ such that $\langle d \rangle_1 + \ldots + \langle d \rangle_p \equiv d \mod 2$.

Arithmetic secret sharing

Boolean secret sharing is extended to Additive secret sharing by representing functions as arithmetic circuits: Inputs are additively shared for the operations, in particular addition and multiplication. In other words, arithmetic secret sharing allows two (or multi) parties to compute additions and/or multiplications over additively shared values in $\mod 2^\ell$ where ℓ denotes the bit size for the additive secret sharing. Addition and multiplication gates correspond to XOR and AND gates, respectively. The addition over the shared values is computed locally whereas performing the multiplication operation requires two parties to interact. This can be performed with Beaver's multiplication triplets [39].

We denote $\langle . \rangle_k$, $k = 1, .., p$, to represent an Arithmetic secret share. A secret input d is split into p shares, using $\mathrm{AS}.\mathrm{Share}(p, d)$ which returns $\langle d \rangle_1, \ldots, \langle d \rangle_p$ such that $\langle d \rangle_1 + \ldots + \langle d \rangle_p \equiv d \mod 2^\ell$.

3.2.4 Available Libraries

Several open-source implementations have been proposed to make 2PC/MPC systems practical in recent years. Some [40, 41] include high-level program description languages and corresponding compilers used to specify a function to securely compute and translate it into some Boolean or arithmetic secret sharings ($\mathrm{BS}.\mathrm{Share}$ or $\mathrm{AS}.\mathrm{Share}$) while some solutions [42–44] propose more comprehensive frameworks consisting of libraries, languages and their compilers, runtime environments and OT tools. Some examples supporting 2PC

are FairPlay[1], JustGarble[2], ABY[3], TinyGarble[4] while some support implementations for MPC such as FairPlayMP[5], SCAPI[6], SCALE-MAMBA[7], MP-SPDZ[8], MOTION[9], etc.

3.3 Homomorphic Encryption

Homomorphic encryption (HE) is a breakthrough cryptographic technique that allows an untrusted third party such as a cloud server to perform the addition and/or multiplication operations over ciphertexts without the need for decrypting or having access to the underlying or resulting data [45–47]. HE is initially proposed by Rivest, Adleman, and Dertouzos in 1978 [48] and further developed by Gentry in 2009 [49, 50]. We denote $[.]_i$ to represent an HE ciphertext encrypted using the key of party i.

A typical scenario can be described as follows: The party Alice can delegate some of her computations over sensitive data to an untrusted third party, Bob. Alice encrypts her data with her public key (in green in Figure 3.2) and sends it to Bob. When receiving the data, Bob can evaluate a function over Alice's encrypted inputs and obtain the encrypted result. Bob sends the result back to Alice who is the only party able to decrypt it with her private (or secret) key (in red in Figure 3.2). Note that a HE scheme can be either symmetric or asymmetric; in this example, we illustrate the asymmetric one.

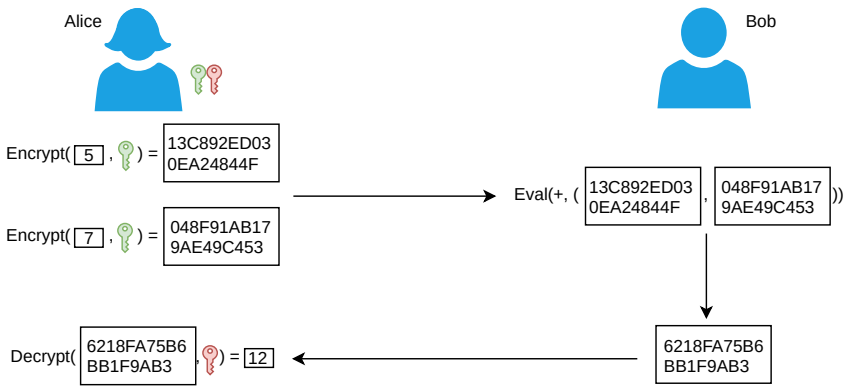


Figure 3.2 – Illustration of asymmetric Homomorphic Encryption

Formally, a (an asymmetric) homomorphic encryption scheme is defined by the following five probabilistic polynomial-time (PPT) algorithms:

- $\mathsf{pp} \leftarrow \text{Setup}(1^\kappa)$: Given the security parameter κ, this algorithm outputs public

[1] https://www.cs.huji.ac.il/project/Fairplay/Fairplay.html
[2] https://github.com/irdan/justGarble
[3] https://github.com/encryptogroup/ABY
[4] https://github.com/esonghori/TinyGarble
[5] https://github.com/FaiplayMP/FairplayMP
[6] https://github.com/cryptobiu/libscapi
[7] https://github.com/KULeuven-COSIC/SCALE-MAMBA
[8] https://github.com/bristolcrypto/SPDZ-2
[9] https://github.com/encryptogroup/MOTION

parameters pp.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{KeyGen}(\mathsf{pp})$: This randomised algorithm takes the security parameter κ as an input and outputs the public-secret keys pair $(\mathsf{pk}, \mathsf{sk})$.
- $\text{ct} \leftarrow \text{Encrypt}(\text{m}, \mathsf{pk})$: This randomised algorithm uses public key pk to transform message m into ciphertext ct.
- $\text{ct}^* \leftarrow \text{Eval}(\mathcal{C}, (\text{ct}_1, \ldots, \text{ct}_l))$: The evaluation algorithm takes a circuit $\mathcal{C}$ and ciphertexts $\text{ct}_1, \ldots \text{ct}_l$ such that $\text{ct}_i = \text{Encrypt}(\text{m}_i, \mathsf{pk})$ and outputs a ciphertext $\text{ct}^* = \text{Eval}(\mathcal{C}, (\text{ct}_1, \ldots, \text{ct}_l))$.
- $\text{m}' \leftarrow \text{Decrypt}(\text{ct}', \mathsf{sk})$: The decryption algorithm uses secret key sk and a ciphertext ct' to output a decrypted message m'.

A symmetric HE can be defined by the same algorithms: The randomised KeyGen algorithm outputs only sk; the randomised Encrypt uses sk to encrypt the plaintext m; and the deterministic Decrypt algorithm finally utilises sk to output the plaintext m'.

An HE scheme should achieve the following properties [51]: (i) The *correctness* property whereby an HE scheme correctly decrypts both ciphertexts computed by the Encrypt algorithm and ciphertexts performed by the Eval algorithm; (ii) the *compactness* property, which ensures the decrypted data evaluated by the Eval algorithm in the message space, where all the defined plaintexts lie; and (iii) the *circuit privacy* property that guarantees the results of the Eval algorithm of some circuit C over some ciphertexts $\text{ct}^1, \ldots, \text{ct}^l$ and the Encrypt algorithm of the result of the same circuit C over the plaintexts which are corresponding the ciphertexts $\text{ct}^1, \ldots, \text{ct}^l$.

HE schemes can be categorised into three groups when considering their operations or the number of operations supported over the encrypted data (i.e., which the circuits HE schemes can evaluate):

3.3.1 Partially Homomorphic Encryption

A cryptosystem that enables a single operation (addition or multiplication) over encrypted data is called a *partially homomorphic encryption* (PHE) scheme [46]. When a PHE scheme enables additions over ciphertexts, it is called additively homomorphic [9, 52], and a PHE scheme that provides multiplications is defined as multiplicatively homomorphic encryption scheme [52, 53].

In this thesis, we utilise an additively homomorphic encryption scheme, namely the Paillier cryptosystem and hence describe it as follows:

The Paillier cryptosystem

Pascal Paillier presents a probabilistic encryption scheme [9] based on composite residuosity problem, which searches an integer x satisfying $x^n = a \mod n^2$ for a given integer a. The Paillier cryptosystem supports additive homomorphism.

The semantically secure Paillier cryptosystem is defined by the following algorithms:

- $\mathsf{pp} \leftarrow \text{Setup}(1^\kappa)$: Given the security parameter κ, this algorithm outputs public parameters pp.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{KeyGen}(\mathsf{pp})$: This randomised algorithm takes the security parameter λ as an input and outputs the public-secret keys pair $(\mathsf{pk}, \mathsf{sk})$ such that $\mathsf{pk} = (N, g)$, where N is the product of two large primes p and q, and $g \in \mathbb{Z}^*_{N^2}$, and the private key is $\mathsf{sk} = (\lambda, \mu)$, where $\lambda = lcm(p-1, q-1)$ and $\mu = (L(g^\lambda \mod N^2))^{-1} \mod N$.
- $\text{ct} \leftarrow \text{Encrypt}(\text{m}, \mathsf{pk})$: This probabilistic algorithm uses public key pk to transform message $\text{m} \in \mathbb{Z}_N$ on modulus N into ciphertext $\text{ct} = g^\text{m} \cdot r^N \mod N^2$, where $r \in \mathbb{Z}^*_{N^2}$.
- $\text{ct}^* \leftarrow \text{Eval}(\mathcal{C}, (\text{ct}^1, \ldots, \text{ct}^l))$: This algorithm takes circuit $\mathcal{C}$ that can be addition and/or scalar multiplication, and ciphertexts $\text{ct}^1, \ldots, \text{ct}^l$ such that $\text{ct}^i = \text{Encrypt}(\text{m}^i, \mathsf{pk})$ and outputs a ciphertext $\text{ct}^* = \text{Eval}(\mathcal{C}, (\text{ct}^1, \ldots, \text{ct}^l))$.
- $\text{m}' \leftarrow \text{Decrypt}(\text{ct}', \mathsf{sk})$: The decryption algorithm uses secret key sk and ciphertext ct' to output decrypted message $\text{m}' = L(\text{ct}'^\lambda \mod N^2) \cdot \mu \mod N$.

As described in [54], the decryption function of the Paillier cryptosystem supports threshold decryption. In our solution, namely SwaNN (see Section 6.3), we use a 2-out-of-2 variant of the threshold decryption which distributes the private key among two parties. The Decrypt algorithm requires both parties to compute the decryption function.

3.3.2 Somewhat Homomorphic Encryption

A cryptosystem whereby we can perform a limited number of both additions or multiplications is named a somewhat homomorphic encryption (SwHE) scheme [45]. The SwHE cryptosystems usually support an unlimited number of additions, but a limited number of multiplications. In 2005, the Boneh-Goh-Nissim (BGN) encryption scheme [55] was proposed where one can perform an arbitrary number of additions and one single multiplication. Later, some cryptosystems are introduced to the state-of-the-art such as the Polly Cracker scheme (both operations are supported, but due to the homomorphic operations, the ciphertext size grows exponentially) [56] or a Polly Cracker with Noise cryptosystem (Additions over ciphertexts do not increase the size of the resulting ciphertext whereas multiplications double the size) [57].

3.3.3 Fully Homomorphic Encryption

Before defining Fully Homomorphic Encryption, we define Levelled Homomorphic Encryption: When an HE scheme supports both additive and multiplicative homomorphisms over encrypted data, and takes an additional input d, namely the maximum depth of circuits containing additions and multiplications which it can evaluate, for the KeyGen algorithm, it is named a Levelled Homomorphic Encryption (LHE) scheme.

With each homomorphic operation (especially multiplications), the noise level in the LHE-encrypted ciphertext increases, and if this noise level exceeds some threshold, the

underlying ciphertext can no longer be correctly decrypted. Therefore, Craig Gentry [50] proposes the first fully homomorphic encryption (FHE) and uses an additional technique, namely bootstrapping, to reduce the noise growth in the ciphertext originating from computations over ciphertexts. With this aim, LHE schemes are used since they can homomorphically evaluate their own Decrypt algorithm (thus, they are called bootstrappable): The resulting noisy ciphertext can be homomorphically decrypted using the encrypted secret key, and further, it is encrypted using a different (or the same) public key to get a "fresh" ciphertext. Therefore, LHE schemes without a depth restriction (or a multiplicative level) are called the FHE schemes. Note that the bootstrapping procedure increases the computational and memory costs of FHE and can be thus considered as a bottleneck for the practical use of FHE.

To decrease the computational time over ciphertexts, Smart et al. [58] propose to encode several plaintexts into a single plaintext to perform them in batches without bringing any additional cost. This procedure is called Single Instruction Multiple Data (SIMD). Remark that SIMD can also be applicable for secure 2PC/MPC to enhance the time for function evaluations.

The research on FHE has been continuing and thus new cryptosystems have been proposed based on Gentry's creative idea to improve its efficiency. As the result of the research, a new HE cryptosystem, BGV, proposed by Brakerski et al. [59], and Fan-Vercauteren [60] in 2012. In 2013, Gentry-Sahai-Waters [61] propose a new scheme having a different way to represent keys, but it is not compatible with existing optimization due to the key representation. Although all these proposed HE schemes are dedicated to integers, the cryptosystem proposed by Cheon et al. [62] allowing to care out computations on floating point numbers was introduced. Recently, the new generation has been introduced with TFHE [63, 64] allowing to have the advantage of previous ones. These presented FHE schemes, namely BFV, CKKS, TFHE, etc., are based on the Learning with Errors (LWE) (or Ring Learning With Error (RLWE)) problem. Authors in [65] prove that both Approximate Greatest Common Divisor problem and LWE problems are equivalent.

In this thesis, we utilise the following fully homomorphic encryption schemes:

The BFV cryptosystem

BFV is an FHE scheme based on RLWE [60, 66]. The semantically secure BFV cryptosystem is defined by the following algorithms:

- $\mathsf{pp} \leftarrow \text{Setup}(1^\kappa)$: Given the security parameter κ, this algorithm outputs public parameters pp.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{KeyGen}(\mathsf{pp})$: This randomised algorithm takes the security parameter κ as an input and outputs the public-secret keys pair $(\mathsf{pk}, \mathsf{sk})$ where $\mathsf{sk} = s \xleftarrow{\$} R_2$, where R_2 is the Ring $R \mod 2$ and $\mathsf{pk} = ([-(a.s+e)]_q, a)$, where $a \xleftarrow{\$} R_q$ where $q = q(\kappa) \geq 2$ is an integer, and $e \xleftarrow{\$} \mathcal{X} = \mathcal{X}(\kappa)$.
- $\text{ct} \leftarrow \text{Encrypt}(\text{m}, \mathsf{pk})$: This probabilistic algorithm uses public key pk to transform

message m into ciphertext $\text{ct} = \Big([p_0.u + e_1 + \Delta.\,\text{m}]_q, [p_1.u + e_2]_q\Big)$ where $u \xleftarrow{\$} R_2$, $e_1, e_2 \xleftarrow{\$} \mathcal{X}$, and $\Delta = \lfloor q/t \rfloor$.

- $\text{ct}^* \leftarrow \text{Eval}(\mathcal{C}, (\text{ct}^1, \ldots, \text{ct}^l))$: This algorithm takes circuit $\mathcal{C}$ that can be a set of addition and/or multiplication, and ciphertexts $\text{ct}^1, \ldots, \text{ct}^l$ such that $\text{ct}^i = \text{Encrypt}(\text{m}^i, \mathsf{pk})$ and outputs a ciphertext $\text{ct}^* = \text{Eval}(\mathcal{C}, (\text{ct}^1, \ldots, \text{ct}^l))$.
- $\text{m}' \leftarrow \text{Decrypt}(\text{ct}', \mathsf{sk})$: The decryption algorithm uses secret key sk and ciphertext ct' to output decrypted message $\text{m}' = \left[\left\lfloor \frac{t.[c_0 + c_1.s]_q}{q} \right\rfloor \right]_t$ where $c_0 = \text{ct}'[0]$ and $c_1 = \text{ct}'[1]$.

The CKKS cryptosystem

CKKS [62] is a FHE scheme based on the BGV scheme [59]. The semantically secure CKKS cryptosystem is defined by the following algorithms:

- $\mathsf{pp} \leftarrow \text{Setup}(1^\kappa)$: Given the security parameter κ, this algorithm outputs public parameters pp.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{KeyGen}(\mathsf{pp})$: This randomised algorithm takes the security parameter κ as an input and outputs the public-secret keys pair $(\mathsf{pk}, \mathsf{sk})$ where $\mathsf{sk} = (1, s)$ such that $s \xleftarrow{\$} R_L$ and $\mathsf{pk} = (b, a)$ such that $b = -a.s + e \mod q_L$ where $e \xleftarrow{\$} \mathcal{DG}(\sigma^2)$[10], and a real value $\sigma = \sigma(\kappa, q_L)$.
- $\text{ct} \leftarrow \text{Encrypt}(\text{m}, \mathsf{pk})$: This randomised algorithm uses public key pk to transform message m into ciphertext $\text{ct} = (ct_0, ct_1) = \nu.pk + m + e_0, e_1) \mod q_L \leftarrow \text{Encrypt}(\text{m}, \mathsf{pk})$ where $\nu \xleftarrow{\$} \mathcal{ZO}(0.5)$[11] and $e_0, e_1 \xleftarrow{\$} \mathcal{DG}(\sigma^2)$.
- $\text{ct}^* \leftarrow \text{Eval}(\mathcal{C}, (\text{ct}^1, \ldots, \text{ct}^l))$: This algorithm takes a circuit $\mathcal{C}$ that can be a set of addition and/or multiplication, and ciphertexts $\text{ct}^1, \ldots, \text{ct}^l$ such that $\text{ct}^i = \text{Encrypt}(\text{m}^i, \mathsf{pk})$ and outputs ciphertext $\text{ct}^* = \text{Eval}(\mathcal{C}, (\text{ct}^1, \ldots, \text{ct}^l))$.
- $\text{m}' \leftarrow \text{Decrypt}(\text{ct}', \mathsf{sk})$: The decryption algorithm uses secret key sk and ciphertext ct' to output decrypted message $\text{m}' = ct'_1 + ct'_0.s \mod q_l$.

The TFHE cryptosystem

Chillotti et al. propose TFHE, a symmetric FHE scheme based on the torus variant of the LWE [63, 64, 67]. The semantically secure TFHE cryptosystem is defined by the following algorithms:

- $\mathsf{pp} \leftarrow \text{Setup}(1^\kappa)$: Given the security parameter κ, this algorithm outputs public parameters pp.

[10] $\mathcal{DG}(\sigma^2)$ stands for the discrete Gaussian distribution of variance σ^2.

[11] $\mathcal{ZO}(\rho)$ is a distribution in which each entry of the vector from $\{0, \pm 1\}^N$ with a probability ρ for being -1 and +1, and a probability $1 - \rho$ for being 0.

- $(\mathsf{sk}) \leftarrow \text{KeyGen}(\mathsf{pp})$: This randomised algorithm takes the security parameter κ as an input and outputs the public-secret keys pair (sk) where $\mathsf{sk} = s \xleftarrow{\$} \{0,1\}^n$.
- $\text{ct} \leftarrow \text{Encrypt}(\text{m}, \mathsf{sk})$: This randomised algorithm uses the public key sk to transform message m into ciphertext $\text{ct} = (a, b)$ where $b = a.s + \varphi \mod 1$, $a \xleftarrow{\$} \mathbb{T}^n$, and $\varphi = \text{m} +$ Gaussian Error of parameter α.
- $\text{ct}^* \leftarrow \text{Eval}(\mathcal{C}, (\text{ct}_1, \ldots, \text{ct}_l))$: This algorithm takes circuit $\mathcal{C}$ that can be a set of addition and/or multiplication, and ciphertexts $\text{ct}_1, \ldots \text{ct}_l$ such that $\text{ct}_i = \text{Encrypt}(\text{m}_i, \mathsf{pk})$ and outputs ciphertext $\text{ct}^* = \text{Eval}(\mathcal{C}, (\text{ct}_1, \ldots, \text{ct}_l))$.
- $\text{m}' \leftarrow \text{Decrypt}(\text{ct}', \mathsf{sk})$: The decryption algorithm uses secret key sk and ciphertext ct' to output decrypted message m' is the nearest message in the message space by rounding $\varphi \leftarrow \text{Decrypt}(\text{ct}', \mathsf{sk}) = b - s.a$.

3.3.4 Available Libraries

There exist several open-source libraries for the implementation of homomorphic encryption schemes, mostly FHE schemes. Some HE libraries can be listed as follows: A C++ implementation SEAL[12] supports BFV and CKKS; PALISADE[13] supports BGV, BFV, CKKS, FHEW, and TFHE based on C++; HElib [14] employs C++ to implement BGV and CKKS; TFHE[15] provides TFHE utilising the C/C++ languages; and lastly, Pyfhel[16] builds its implementation on top of SEAL and PALISADE using the Python language.

3.4 Proxy Re-encryption

A Proxy Re-Encryption (PRE) scheme allows an untrusted third party such as a cloud server to transform a ciphertext between keys without having access to both the secret keys and the cleartext. In a typical scenario of PRE depicted in Figure 3.3, a proxy converts a ciphertext encrypted under Alice's public key for Bob with the help of a given key, namely the re-encryption key $\mathsf{rek}_{A \to B}$. Informally, consider a scenario whereby Alice receives encrypted emails under her public key from several clients. When she leaves for vacation, she wants to authorise Bob to read her emails without sharing her secret key with Bob. Proxy Re-encryption can be useful in the case of when Alice is not available, and on behalf of Alice, Bob could read these emails without the need for having her secret key.

Formally, a proxy re-encryption scheme is defined by the following six algorithms:

- $\mathsf{pp} \leftarrow \text{PRE}.\text{Setup}(1^\kappa)$: Given the security parameter κ, this algorithm outputs public parameters pp.

[12] https://github.com/microsoft/SEAL
[13] https://gitlab.com/palisade/palisade-release
[14] https://github.com/shaih/HElib
[15] https://github.com/tfhe/tfhe
[16] https://github.com/ibarrond/Pyfhel

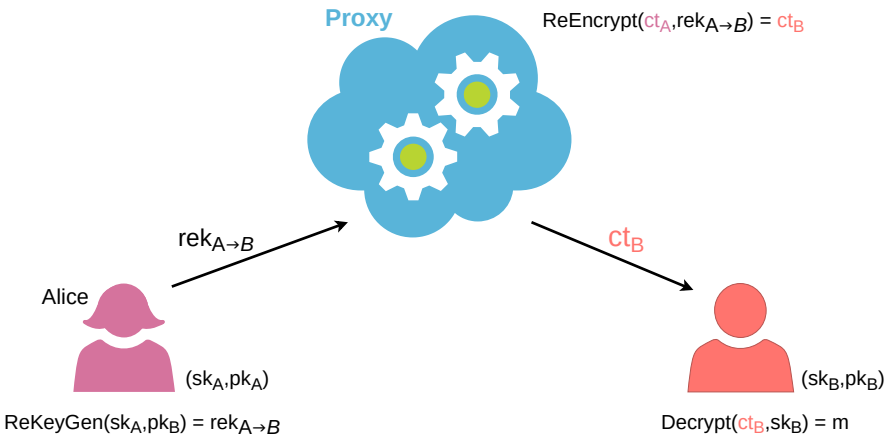


Figure 3.3 – Illustration of Proxy Re-Encryption

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{PRE}.\mathrm{KeyGen}(\mathsf{pp})$: This randomised key generation algorithm takes an input as the given security parameter κ as an input and outputs the public-secret keys pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{rek}_{i \to j} \leftarrow \mathrm{PRE}.\mathrm{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j)$: This re-encryption key generation procedure takes public key pk_j and secret key sk_i from parties P_i and P_j as inputs, and further outputs a re-encryption key, $\mathsf{rek}_{i \to j}$.
- $\mathrm{ct}_i \leftarrow \mathrm{PRE}.\mathrm{Encrypt}(\mathrm{m}, \mathsf{pk}_i)$: The randomised encryption algorithm uses public key pk_i to transform message m to ct_i.
- $\mathrm{ct}_j \leftarrow \mathrm{PRE}.\mathrm{ReEncrypt}(\mathrm{ct}_i, \mathsf{rek}_{i \to j})$: The proxy re-encryption algorithm, namely ReEncrypt (or Π), uses re-encryption key $\mathsf{rek}_{i \to j}$ to transform ciphertext $\mathrm{ct}_i = \mathrm{Encrypt}(\mathrm{m}, \mathsf{pk}_i)$ into ciphertext ct_j.
- $\mathrm{m} \leftarrow \mathrm{PRE}.\mathrm{Decrypt}(\mathrm{ct}_j, \mathsf{sk}_j)$: The decryption algorithm uses secret key sk_j to decrypt ciphertext ct_j and obtain message m.

The concept of PRE is firstly introduced by Blaze et al. [68] as atomic proxy re-encryption utilising the ElGamal encryption scheme and the Fiat-Shamir signature scheme. Further, Ivan and Dodis [69] generalise this concept with strong security assumptions, as well. With this scheme, some standard cryptographic scheme such as ElGamal [52], RSA [53] or RSA Hash-and-Sign [70, 71] can be transformed into a proxy function.

The early proposed PRE such as [68] suffers from the following shortcomings: Once the proxy re-encrypts the ciphertext of Alice to Bob with re-encryption key $rk_{A \to B}$, Alice is still able to decrypt the re-encrypted message. This is defined as the bidirectionality property in [72]. This scheme is also vulnerable against a collusion between the proxy and one of the two parties. Thus, the scheme considers the proxy fully trusted. To cope with these weaknesses, Ateniese et al. [73] introduce a unidirectional, collusion-resistant, and CPA-secure PRE scheme based on the decisional bilinear Diffie-Hellman (DBDH)

assumption [74]. Moreover, in [75], authors propose an alternative scheme that is secure against chosen ciphertext attacks.

3.4.1 Homomorphic Proxy Re-encryption

As its name indicates, a Homomorphic Proxy Re-encryption (H-PRE) is the combination of two cryptographic constructions: (i) homomorphic encryption (HE) which enables operations over encrypted data (See Section 3.3); and (ii) proxy re-encryption (PRE) which allows a third-party proxy such as a cloud server to transform ciphertexts encrypted with a public key into ciphertexts encrypted with another public key without learning any information on the plaintext (See Section 3.4). Recently several works [76–82] studied the combination of PRE and HE. We define such schemes as H-PRE schemes: Some existing H-PRE are based on the Paillier cryptosystem [9] whereas some others are based on their newly designed HE; and one study [80] proposes two examples of H-PRE, namely BV-PRE and NTRU-ABD-PRE.

A scenario for H-PRE illustrated in Figure 3.4 can be described as follows: Several clients want to send their data to Alice, who computes some operations such as the sum over these data. Their data are very sensitive, and clients, therefore, encrypt their data using Alice's public key and send them to her. Unfortunately, Alice is not available and delegates to Bob the computation of the sum over the received data. Alice generates the re-encryption key $\mathsf{rek}_{A\to B}$ using her secret key and Bob's public key and sends it to the proxy, seen as a cloud server. When the data are sent to Alice, the proxy re-encrypts them for Bob, computes the sum over them, and further sends the resulting sum to Bob. When the sum is received, Bob can decrypt it by using his own secret key.

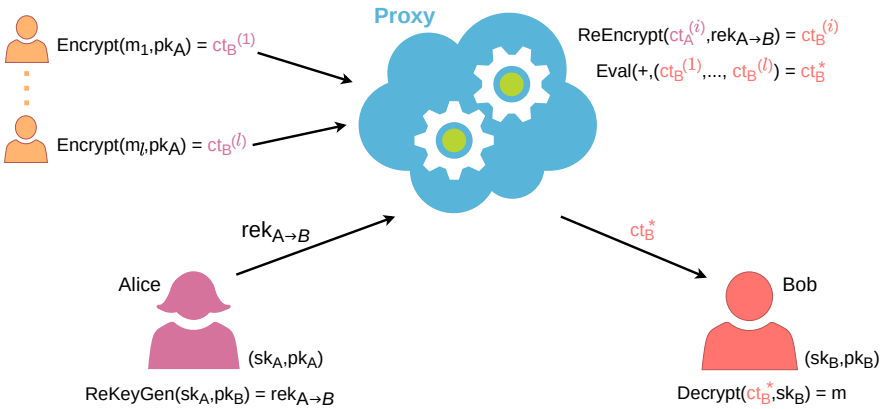


Figure 3.4 – Illustration of Homomorphic Proxy Re-Encryption

More formally, a H-PRE scheme consists of the following seven polynomial-time algorithms:

- $\mathsf{pp} \leftarrow \text{H-PRE}.\,\text{Setup}(1^\kappa)$: Given the security parameter κ, this algorithm outputs public parameters pp.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow$ H-PRE . KeyGen(pp): This probabilistic key generation algorithm takes the security parameter κ as input, and outputs a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{rek}_{i \to j} \leftarrow$ H-PRE . ReKeyGen($\mathsf{sk}_i, \mathsf{pk}_j$): This re-encryption key generation algorithm takes secret key sk_i and public key pk_j as inputs and returns re-encryption key, $\mathsf{rek}_{i \to j}$. This re-encryption key will further allow the proxy to re-encrypt a message encrypted with pk_i to a message that can be decrypted by the party holding secret key sk_j corresponding to public key pk_j.
- ct $\leftarrow$ H-PRE . Encrypt(m, pk): This randomised algorithm encrypts message m using public key pk and returns the resulting ciphertext ct.
- $\mathrm{ct}_j \leftarrow$ H-PRE . ReEncrypt($\mathsf{rek}_{i \to j}, \mathrm{ct}_i$): This re-encryption algorithm transforms ciphertext ct_i into ciphertext ct_j using re-encryption key $\mathsf{rek}_{i \to j}$.
- $\mathrm{ct}^* \leftarrow$ H-PRE . Eval($\mathcal{C}, (\mathrm{ct}^{(1)}, \cdots, \mathrm{ct}^{(l)})$): Given circuit $\mathcal{C}$ and ciphertexts $\mathrm{ct}^{(1)}, \cdots, \mathrm{ct}^{(l)}$ where $\mathrm{ct}^{(i)} =$ H-PRE . Encrypt($\mathrm{m}^{(i)}, \mathsf{pk}_j$), the algorithm outputs ciphertext ct^* which corresponds to the encrypted evaluation of $\mathcal{C}$ over ciphertexts $\mathrm{ct}^{(1)}, \cdots, \mathrm{ct}^{(l)}$.
- $\mathrm{m}' \leftarrow$ H-PRE . Decrypt($\mathsf{sk}, \mathrm{ct}^*$): This algorithm decrypts the received ciphertext ct^* using secret key sk and outputs the plaintext m'.

According to [49, 83], any FHE scheme can be transformed into a H-PRE scheme and its correctness implies the correctness of the resulting H-PRE. Furthermore, the same studies show that a H-PRE scheme is semantically (or indistinguishability under chosen-plaintext attack (IND-CPA)) secure if the underlying HE scheme is semantically secure. We note that some H-PRE schemes such as BFV-PRE are implemented in PALISADE [84].

3.5 Multi-key Fully Homomorphic Encryption

In this thesis, we present a particular homomorphic encryption scheme that enables one or multiple parties to encrypt or decrypt data under multiple keys. Two different multi-key homomorphic encryption schemes have been proposed by the state-of-the-art namely: (i) Asymmetric multi-key FHE whereby each party encrypts its data with its own public key and some homomorphic operations are performed over multiple data encrypted with multiple keys [85–87]; and (ii) symmetric multi-key FHE which is similar to the previous scheme except that the encryption key is symmetric [88].

A typical scenario can be described with the following example: Several data owners like DO_1, DO_2, and DO_3 can delegate some computations over their sensitive data to an untrusted third party, a cloud server. Each data owner DO_i encrypts its data with its public key pk_i (in green in Figure 3.5) and sends it to the cloud server. The cloud server can evaluate a function (sum of them in this example) over each DO_i's encrypted input $[\mathbf{X}]_i$ to obtain the encrypted result $[\mathbf{Y}]_{1,2,3}$ thanks to the multi-key FHE. The cloud server sends the result to them to partially decrypt using their secret key (in red in Figure 3.5).

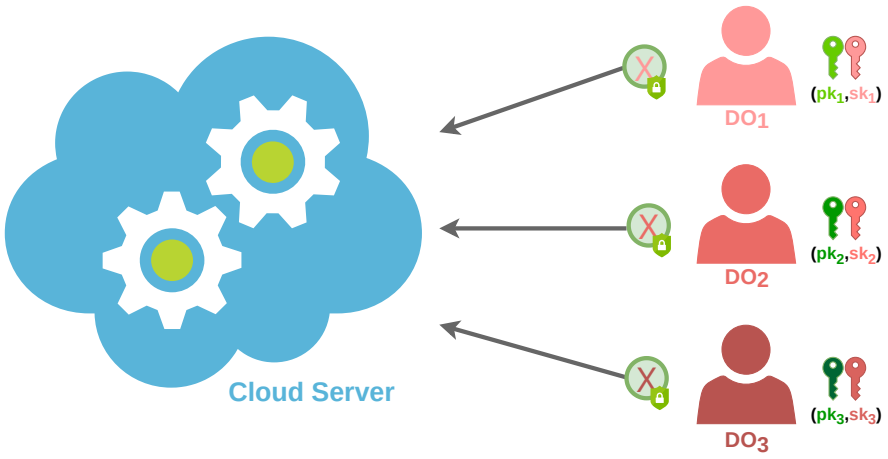


Figure 3.5 – Illustration of Multi-key Fully Homomorphic Encryption

Further, each DO_i sends the partial decrypted result to other DO_j and merges all partial decrypted results to obtain the actual result. Note that a multi-key FHE scheme can be either symmetric or asymmetric; we illustrate the asymmetric one in this example.

3.5.1 Asymmetric Multi-key Fully Homomorphic Encryption

Chen et al. [87] propose semantically secure, multi-key variants of BFV [60, 66] and CKKS [62]. Basically, multiple parties individually encrypt their private data using their own public key, and further homomorphic operations are performed over all these data. All parties contribute to the decryption of the output.

An asymmetric multi-key fully homomorphic encryption (MK-FHE) scheme is defined by seven PPT algorithms.

- $\mathsf{pp} \leftarrow \text{MK-FHE}.\text{Setup}(1^\kappa)$: Given the security parameter κ, this algorithm outputs public parameters pp.
- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \text{MK-FHE}.\text{KeyGen}(\mathsf{pp})$: This randomised algorithm takes the public parameters pp as input and outputs a pair of secret and public keys $(\mathsf{sk}, \mathsf{pk})$.
- $\mathrm{ct} \leftarrow \text{MK-FHE}.\text{Encrypt}(\mathrm{m}, \mathsf{pk})$: This randomised algorithm encrypts message m with public key pk and returns ciphertext $\mathrm{ct} = (c_0, c_1)$.
- $(\mathrm{ct}^*, T^*) \leftarrow \text{MK-FHE}.\text{Pre-process}(\bar{\mathrm{ct}} = (c_0, \ldots, c_{k_i}), T = \{id_1, \ldots, id_{k_i}\})$: This algorithm basically extends the input ciphertext with additional 0's in order to be able to perform the homomorphic operation over all the underlying k keys according to the mapping defined in T^*. Hence, the algorithm returns $\mathrm{ct}^* = (c_0^*, \ldots, c_k^*)$ such that $c_0^* = c_0$ and $c_i^* = \begin{cases} c_j & \text{if } i = id_j \text{ for some } 1 \leq j \leq k_i, \\ 0 & \text{otherwise.} \end{cases}$

- $\mathrm{ct}^* \leftarrow \text{MK-FHE}.\mathrm{Eval}(\mathcal{C}, (\mathrm{ct}_1^*, \ldots, \mathrm{ct}_l^*))$: This algorithm evaluates circuit $\mathcal{C}$ over the l ciphertexts encrypted with multiple keys.
- $\mu_i \leftarrow \text{MK-FHE}.\mathrm{PartialDecrypt}(c_i^*, \mathsf{sk}_{id_i})$: This algorithm takes as input c_i^* from ciphertext ct^* corresponding to the party holding secret key sk_{id_i} and outputs a partially decrypted information μ_i.
- $\mathrm{m}' \leftarrow \text{MK-FHE}.\mathrm{Merge}(c_0^*, \mu_1, \ldots, \mu_k)$: This algorithm takes as input all the partial decryptions derived from a ciphertext ct^* and outputs the final plaintext m'.

3.5.2 Symmetric Multi-key Fully Homomorphic Encryption

Chen et al. [88] propose the multi-key version of TFHE, namely MK-TFHE, whereby each party uses its symmetric secret key to encrypt and decrypt the data. Similar to MK-FHE, the MK-TFHE scheme is defined by seven PPT algorithms:

- $\mathsf{pp} \leftarrow \text{MK-TFHE}.\mathrm{Setup}(1^\kappa)$: This algorithm outputs public parameters pp given security parameter κ.
- $\mathsf{sk} \leftarrow \text{MK-TFHE}.\mathrm{KeyGen}(\mathsf{pp})$: This algorithm randomly generates secret key sk given the public parameters pp.
- $\mathrm{ct} \leftarrow \text{MK-TFHE}.\mathrm{Encrypt}(\mathrm{m}, \mathsf{sk})$: This randomised algorithm encrypts message m with secret key sk and outputs ciphertext $\mathrm{ct} = (b, a)$.
- $(\mathrm{ct}^*, T^*) \leftarrow \text{MK-TFHE}.\text{Pre-process}(\bar{\mathrm{ct}} = (b, a_1, \ldots, a_{k_i}), T = \{id_1, \ldots, id_{k_i}\})$: Similar to the case of MK-FHE, this algorithm basically extends the input ciphertext with additional 0's in order to be able to perform the homomorphic operation over all the underlying k keys according to the mapping defined in T^*. Hence, the algorithm returns $\mathrm{ct}^* = (b^*, a_1^* \ldots, a_k^*)$ such that $b^* = b$ and $a_i^* = \begin{cases} a_j & \text{if } i = id_j \text{ for some } 1 \leq j \leq k_i, \\ 0 & \text{otherwise.} \end{cases}$
- $\mathrm{ct}^* \leftarrow \text{MK-TFHE}.\mathrm{Eval}(\mathcal{C}, (\mathrm{ct}_1^*, \ldots, \mathrm{ct}_l^*))$: This algorithm evaluates circuit $\mathcal{C}$ over the l ciphertexts encrypted with multiple keys.
- $\mu_i \leftarrow \text{MK-TFHE}.\mathrm{PartialDecrypt}(a_i^*, \mathsf{sk}_{id_i})$: This algorithm takes as input a_i^* from ciphertext ct^* corresponding to the party holding secret key sk_{id_i} and outputs a partially decrypted information μ_i.
- $\mathrm{m}' \leftarrow \text{MK-TFHE}.\mathrm{Merge}(b, \mu_1, \ldots, \mu_k)$: This algorithm takes as input all the partial decryptions derived from a ciphertext ct^* and outputs the final plaintext m'.

MK-TFHE has been implemented as a proof-of-concept in [89], and only the NAND gate is utilised in this library.

3.6 Threshold Fully Homomorphic Encryption

Threshold fully homomorphic encryption (Th-FHE, [90,91]) is a homomorphic encryption scheme whereby multiple parties jointly generate a public-secret key pair using their public and secret keys. The resulting secret key is not disclosed to any party. Each party encrypts its private data using the common public key and operations are performed over encrypted data. In order to decrypt the result, these parties partially decrypt with their individual secret keys and merge the partially decrypted values in a similar manner to MK-FHE.

A typical scenario can be described with the following example: Several data owners like DO_1, DO_2, and DO_3 can delegate some computations over their sensitive data to an untrusted third party, a cloud server. Each data owner DO_i encrypts its data with one unique public key pk (in green in Figure 3.6) and sends it to the cloud server. The cloud server can evaluate a function (sum of them in this example) over each DO_i's encrypted input $[\mathbf{X}]$ to obtain the encrypted result $[\mathbf{Y}]$ thanks to the threshold FHE. The cloud server sends the result to them to partially decrypt using their secret key (in red in Figure 3.6). Further, each DO_i sends the partial decrypted result to other DO_j and merges all partial decrypted results to obtain the actual result.

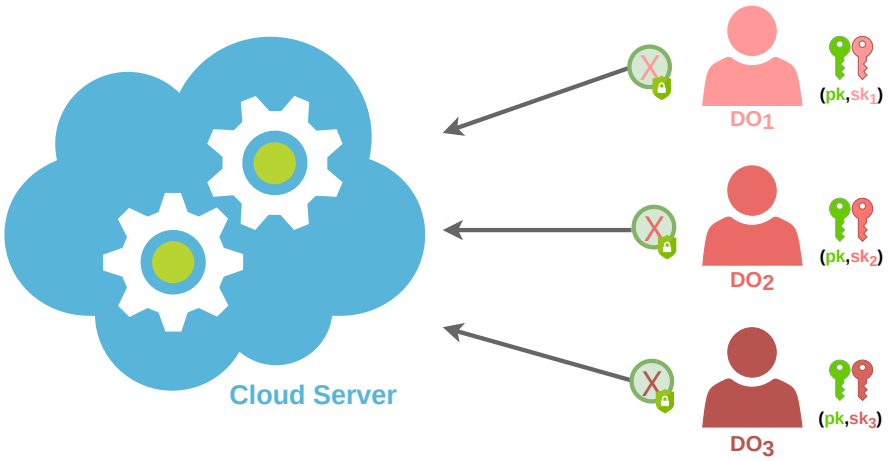


Figure 3.6 – Illustration of Threshold Fully Homomorphic Encryption

A semantically secure Th-FHE scheme contains six PPT algorithms:

- $\mathsf{pp} \leftarrow$ Th-FHE . Setup(1^κ): This algorithm outputs public parameters pp given security parameter κ.
- $(\mathsf{pk}, \mathsf{sk}_{id_1}, \ldots, \mathsf{sk}_{id_k}) \leftarrow$ Th-FHE . KeyGen(pp): This randomised algorithm generates k secret keys sk_{id_i} and common public key pk given the public parameters pp.
- $\mathrm{ct} \leftarrow$ Th-FHE . Encrypt$(\mathrm{m}, \mathsf{pk})$: This randomised algorithm encrypts message m with public key pk and outputs ciphertext ct.

- $\mathrm{ct}^* \leftarrow$ Th-FHE . Eval$(\mathcal{C}, (\mathrm{ct}_1, \ldots, \mathrm{ct}_l))$: This algorithm evaluates circuit $\mathcal{C}$ over the l ciphertexts.
- $\mu_i \leftarrow$ Th-FHE . PartialDecrypt$(\mathrm{ct}^*, \mathsf{sk}_{id_i})$: This algorithm takes as input ciphertext ct^* and the secret key sk_{id_i} of party i and outputs a partially decrypted information μ_i.
- $\mathrm{m}' \leftarrow$ Th-FHE . Merge$(\mu_1, \ldots, \mu_k)$: This algorithm takes as input all the partial decryptions derived from a ciphertext ct^* and outputs the final plaintext m'.

Note that some Th-FHE schemes for BFV and CKKS are implemented in PALISADE [84].

3.7 Hybrid Protocol

Some cryptographic implementations rely on a hybrid method that consists of a mixed protocol by combining the use of arithmetic circuits with homomorphic encryption and/or the use of Boolean circuits. This mixed approach can show better performance results when comparing stand-alone use of 2PC/MPC or HE.

Some open-source libraries have been proposed. For example, TASTY[17] can be counted as a hybrid protocol library based on the use of Yao's GCs and additively HE, and some examples for the Hybrid protocol that utilise mixed 2PC/MPC protocols such as secret sharing and Yao's GCs are ABY, MOTION, etc.

3.8 Summary of Cryptographic techniques

In this chapter, cryptographic techniques we employ in our privacy-preserving MLaaS solutions have been presented, namely secure two-party computation, partially homomorphic encryption, homomorphic proxy re-encryption, multi-key homomorphic encryption, and threshold homomorphic encryption. A PHE scheme is more efficient in consideration of its reasonable computation costs whereas an FHE cryptosystem provides more flexibility to perform any operations. The latter cryptosystem is more expensive in computations than a partially homomorphic one. Furthermore, an HE scheme non-interactively enables nonlinear operations and polynomials in the ML techniques over encrypted data. Unfortunately, it results in high computational costs. However, an MPC/2PC scheme provides more realistic computation performance; yet, it suffers from a higher bandwidth use due to its interactive nature.

[17] https://github.com/tastyproject/tasty

Part I

Privacy-preserving single-server machine learning techniques

Chapter 4

Privacy-preserving Neural Network Classification

Dance first.
Think later.
It's the natural order.

Samuel Beckett

In this chapter, we study and identify the challenges of neural networks when integrating cryptographic techniques. We further review the state-of-the-art and finally introduce our solutions for privacy-preserving neural network classification which are based on different cryptographic techniques: The first solution based on 2PC, the second solution based on PHE and 2PC, and the last solution based on schemes of FHE and PRE, namely H-PRE.

4.1 Introduction

Consider a scenario whereby a client, or *a querier*, would like to classify its input $\mathbf{X}$ utilising a NN model, namely $\mathbf{M}$ in the cloud, and *a cloud server* would classify the underlying input and obtain the corresponding output $\mathbf{Y}$, where $\mathbf{Y} = \mathbf{M}(\mathbf{X})$. The NN model $\mathbf{M}$ is usually trained and constructed over privacy-sensitive data sent by multiple clients, and inputs and outputs for the prediction/classification phase are personal data. They should, therefore, be kept confidential to any parties except their owners. In order to guarantee data privacy, i.e., the privacy of the input $\mathbf{X}$ and the output $\mathbf{Y}$ against the cloud server, and the NN model $\mathbf{M}$ against clients, some cryptographic

technique such as multi(two)-party computation (MPC/2PC) [40, 92] or homomorphic encryption (HE) [9, 49, 50, 59] is employed. Indeed, the protection of data privacy in the NN classification drew the attention of many researchers and several mechanisms that provide privacy protection in neural network predictions are proposed. In these proposals, the main goal is to delegate the classification operations to the cloud server. Therefore, the already trained NN model is possessed by or is outsourced to the cloud server in the cleartext/plaintext form or in the encrypted form.

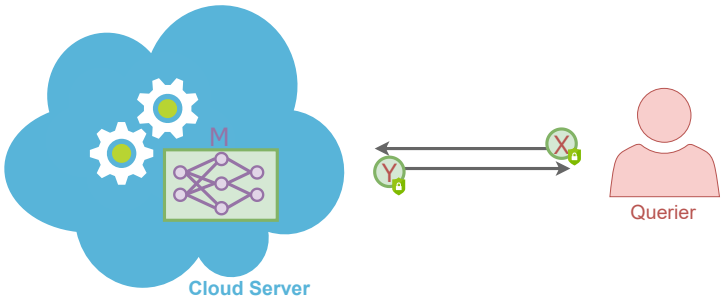


Figure 4.1 – Plaintext model in the single-server scenario

Having studied existing proposals, we design three privacy-preserving NN prediction solutions according to the scenarios, including the trained NN model being in the plaintext or encrypted form in the cloud when ensuring privacy for all the underlying data. While PAC and SwaNN classify input $\mathbf{X}$ over some model $\mathbf{M}$ which is plaintext (i.e. no cryptography is applied) on the cloud, ProteiNN employs a model $\mathbf{M}$ in the encrypted form for the prediction queries. Moreover, in PAC [93], we utilise 2PC to preserve the privacy of the Electro-Cardiogram (ECG) data and SwaNN [94] switches the computations between HE and 2PC during the image (hand-written digit) classification in the previously mentioned scenario whereas ProteiNN [95] extends this scenario that we name the *one-to-one scenario* depicted in Figure 4.1, by enabling *a model provider* to securely outsource its model $\mathbf{M}$ to the cloud server and consider a *one-to-many scenario* in Figure 4.2 whereby *many clients* can query the model. Additionally, while delegating the $\mathbf{M}$ operations to the cloud server, the model provider still wishes to maintain the control over the use of this model by legitimate and authorised clients, only.

4.2 Privacy vs. Neural Network

This section focuses on the neural network classification and its operations and identifies the challenges to design the privacy-preserving variant of it.

As previously presented in Section 2.1, most of the neural network functions such as Cost function, SGD or activation functions consist of complex (or non-linear) operations even for machine learning researchers. The classification phase is much simpler than the training phase regarding the NN phases containing complex operations. Moreover, the need for classifying some inputs using some NN model is more demanded than training a NN model when comparing the number of clients and data-driven companies. Since

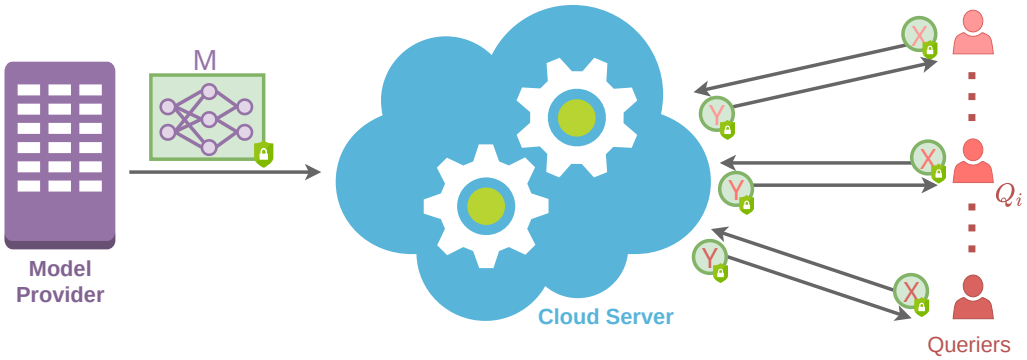


Figure 4.2 – Encrypted model in the single-server scenario

clients would like to take advantage of a cloud server and delegate the NN operations to the cloud server, their inputs need to be protected via some cryptographic technique before their outsourcing. Therefore, since the NN classification has more advantages, we propose to design the neural network classification solutions when considering the integration of the cryptographic technique(s) and their bottlenecks.

The main goal is to allow various clients to apply a neural network (NN) model that belongs to some third party and own their sensitive output; at the same time, the third party learns nothing about data, and the clients learn the classification results only and nothing about the NN model. Hence, we aim at designing a solution where a third party can execute the classification model without leaking and even discovering information about the input data and the output data. On the other hand, the classification model can also be considered as confidential against its clients who will send their queries for the classification task. This model itself can also have some business value and therefore needs to be protected. For this respect, we assume that the model should be unknown to the parties querying it. Performing some operations over data while these are kept confidential requires the use of cryptographic techniques such as HE or MPC/2PC. While the integration of such techniques offers better privacy guarantees, they, unfortunately, introduce some non-negligible overhead in terms of computation and communication. Furthermore, these techniques may not always be compatible with complex NN operations presented in Chapter 2. Therefore, we believe that to design the privacy-preserving variant of the underlying classification technique, the actual classification model should be revisited and built while taking the privacy requirements into consideration, as well. Hence, we propose to follow a *privacy-by-design* approach and consider privacy requirements at the design phase of the neural networks.

The following three challenges are identified when building a NN model customised for the use of cryptographic techniques:

- **Challenge 1: Size of the neural network.** The size of a neural network is defined as the size of the input and output vectors, the number of layers, and the number of neurons in the model. Such parameters have a significant impact on the complexity of the model. In order to efficiently integrate cryptographic techniques, the size

of the neural network should be optimised. Therefore, while designing the neural network compatible with the use of cryptographic techniques, one should reduce the size of the input layer, the hidden layers, and the output layer.

- **Challenge 2: Complex neural network operations.** A neural network involves various operations executed by each neuron during the classification phase. These include sophisticated operations such as `Sigmoid` or `tanh` that existing cryptographic techniques may not easily and efficiently support. Hence, the underlying operations should be optimised and sometimes even transformed into simpler functions such as the `Square` function (i.e., x^2) or some low degree polynomial when designing the privacy-preserving variant of the NN model.
- **Challenge 3: Real numbers instead of integers.** Most of the operations in NN are executed over real numbers whereas cryptographic techniques usually support integers. Therefore, there is a need for either supporting floating point numbers or approximating them to integers. Nevertheless, such an approximation should not have a significant impact on the accuracy of the NN model.

To reduce the overhead resulting from introducing the privacy-preserving variants of the complex NN operations, the number of these operations, hence, the size of the neural network has to be optimised. On the other hand, such optimisations should not have an impact on the actual accuracy of the NN model and its performance evaluation.

4.3 Prior Work

In this section, we study existing privacy-preserving neural networks solutions and regroup them into several categories according to the underlying cryptographic techniques, and further we have proposed a Systematisation of Knowledge (SoK) paper [96] published in *IFIP Summer School on Privacy and Identity Management 2019*.

4.3.1 2PC-based solutions

The first category consists of solutions based on secure two-party computation. MiniONN [97] is the first privacy-preserving neural network protocol based on 2PC between a client and a cloud server: The client and the cloud server additively share each of their input and output values for each layer of the neural network. To ensure data privacy, MiniONN defines oblivious transformations for each CNN operation and implements the transformations using the ABY framework [42]. Furthermore, Rouhani et al. [98] propose DeepSecure which is based on Yao's GCs to securely compute the deep learning model. The authors are able to use *Sigmoid* and *tanh* as activation functions thanks to the optimisation of garbled circuits. Ball et al. [99] propose an extension to DeepSecure that is a secure evaluation of the NN classifier based on garbled circuits. Unlike DeepSecure, authors in [99] make use of arithmetic circuits instead of Boolean circuits and further use some improved techniques for the computation of matrix multiplication, activation function, and `Max` pooling. Another study which uses 2PC is Chameleon by Riazi et

al. [100]. Authors propose a protocol that switches among sharing circuits for secure function evaluation, where the client and the cloud server jointly perform some computation without disclosing their inputs. Chameleon can be considered as an alternative protocol to ABY [42]. TFEncrypted [101] is another framework which enables secure computation in TensorFlow [102] using secret sharing and secure channels between the parties. Similar to Chameleon, EzPC [103] is a cryptographic cost-aware secure 2PC protocol generator and makes use of arithmetic and Boolean circuits for a secure NN classification. XONN [104] is a privacy-preserving binarised NN classification solution utilising Yao's GCs. Dalskov et al. [105] study a quantised NN in presence of malicious adversaries. Recently, CrypTFlow2 [106] proposes a 2PC-based NN classification defining new protocols for ReLU, Max pooling, Argmax, and division.

4.3.2 HE-based solutions

In the second category, we analyse (fully) homomorphic encryption ((F)HE)-based solutions. To the best of our knowledge, CryptoNets [107] is the first privacy-preserving neural network protocol based on FHE and proposes to use x^2 in the activation layer. Authors in [107] use the SEAL library [108] to compute convolutional neural network predictions on encrypted images. Similar to CryptoNets, CryptoDL [109], Chabanne et al. [110] and Ibarrondo et al. [111] use FHE for the design of privacy-preserving neural networks. The main difference of these approaches with CryptoNets is the fact that they approximate nonlinear functions with higher degree polynomials using different techniques such as Taylor series, numerical methods or Chebyshev polynomials. The use of batch normalization is also proposed to obtain some performance gain. The goal of all these solutions is to keep a good level of accuracy while using FHE to protect the input data. Later on, Bourse et al. [112] uses a conversion of a trained neural network to a Discretised Neural Network (DiNN) using TFHE [67]. Authors claim that DiNN can be used for deep neural networks with large number of neurons. Similarly, TAPAS [113] also proposes binary neural networks over TFHE-encrypted data. However, TAPAS differs from [112] mainly due to the ability of the cloud server to update the neural network at any time without the need for the data being re-encrypted by the client. More recent works ([114] and [115]) propose the idea of training neural networks over FHE-encrypted data and classifying encrypted predictions. In these studies, the client supplies the training data in its encrypted form using its own public key, and the cloud server trains this encrypted data to build the encrypted model. This model is further used by its owner to classify a new encrypted input. Because both the training data and the model are encrypted, the cloud server cannot discover any information on both phases. Authors claim to achieve a reasonable performance. Moreover, Faster CryptoNets [116] employs an FHE scheme to protect the neural network model and data. Authors [116] propose some efficient polynomial approximations for activation functions. This study also includes a training phase that uses differential privacy to protect the data. CHET [117], a compiler, creates FHE-based code for privacy-preserving NN classifications. Similarly CHET, nGraph-HE, nGraph-HE2, and MP2ML [118–120] are compilers to run a HE-based inference over TensorFlow.

4.3.3 Hybrid solutions

Few early approaches, such as [121] and [122], also use the Paillier encryption scheme and Yao's GCs. Furthermore, authors in [121] and [122] do not provide any performance results of their solution executed over the encrypted data. Additionally, Gazelle [123] is a secure neural network inference scheme implemented under a dedicated lattice-based additively homomorphic encryption scheme proposed in the paper. This solution also makes use of Yao's GCs to perform ReLU and to reduce the noise in the ciphertext. A recent study, namely DELPHI [124] improves Gazelle by using planner to find the best trade-off between efficiency (using polynomials) and accuracy (using Yao's GCs).

4.3.4 Solutions based on other cryptographic techniques

Finally, there exist several works, such as [125], [126], and [127], which propose to combine some machine learning techniques, including the neural network, with trusted hardware.

A recent work [128] reviews the state-of-the-art solutions and further reproduces them, which have a public source code, to analyse them and discuss their deployment issues if exist.

In the following sections, we introduce three solutions for the privacy-preserving neural networks classification problem. These solutions differ with respect to the underlying cryptographic technique.

4.4 PAC: Privacy-preserving Arrhythmia Classification with neural networks

In this section, we present PAC [93], in which we investigate the use of 2PC for the neural network classification algorithm, and in particular the privacy-preserving arrhythmia classification. This work was published in *FPS 2019, 12th International Symposium on Foundations and Practice of Security* and awarded as "the best paper".

Neural Networks (NN) is the most utilised machine learning technique to support pharmacists and doctors in analysing patients' data and quickly diagnosing a particular disease such as heart arrhythmia that can cause sudden death. Nowadays, this disease can be detected at very early stages with the help of smart wearable devices that can record the heart rate data, which is called as Electro-Cardiogram (ECG). Nevertheless, the ECG data is considered privacy-sensitive. There is an urgent need for cryptographic techniques enabling the protection of such collected data while still performing predictive analytics and improving individuals' lives. These techniques will also help health analytics companies be compliant with GDPR.

In PAC, we aim at addressing privacy concerns raised by the analysis of the ECG data for arrhythmia classification. Our goal is to enable a cloud server to perform arrhythmia classification without discovering the input ECG data to the NN operations. On the other hand, we also look into the problem from the cloud server side as it cares about keeping the design of its NN model confidential from its queriers. Queriers should not be able to discover the details about the underlying NN model. The challenge often manifests as a

choice between the privacy of the querier and the secrecy of the NN model parameters. We propose to reconcile both parties, namely the cloud server and the queriers, and combine the NN technique with 2PC. Since the 2PC protocols cannot efficiently support all NN operations, we propose to revisit the underlying NN operations and design a new, customised NN model that one can execute to classify arrhythmia accurately, and this, without disclosing neither the input ECG data to the cloud server nor the model parameters to the queriers.

4.4.1 Problem Statement

In this section, we introduce the problem of arrhythmia classification and identify the main challenges to ensure the privacy of the ECG data at the same time.

As defined in [129], cardiac arrhythmias are abnormal heart rhythms, which cause the heart to beat too fast (tachycardia) or too slow (bradycardia) and to pump blood less effectively. These irregularities can be detected and classified by analysing the ECG signals of a heart. Doctors classify arrhythmia into several types according to such behaviours of the heart. Therefore, we focus on the classification of heartbeats extracted from the ECG signals into different classes of arrhythmia, making the use of NN: Building the arrhythmia classifier model involves the design of the architecture of NN, such as the number of the layers, the size of the input, the number of neurons in each layer, and the underlying operations that each neuron has to perform, and one should prepare a dataset for the ECG heartbeats representing different arrhythmia types.

ECG signals representing patients' heartbeats are considered as sensitive information. Thus, outsourcing the arrhythmia classifier to cloud servers may put the privacy of the patients at risk. Hence, we propose a solution for the arrhythmia classification problem whereby a user (or a querier) can execute the NN model without leaking and even discovering information about the input data and its output. Moreover, the NN model also has some business value, and therefore it should be unknown to its queriers.

Thanks to the cryptographic techniques (see Chapter 3), one can execute NN operations over data being kept confidential. Although such a technique offers better privacy guarantees, it can result in some non-negligible computation and communication overhead. Moreover, the underlying technique may be efficiently run over the complex NN operations: The design and development of the privacy-preserving NN classification, thus, is a must, and the operations in NN need to be revisited when considering the privacy requirements. The revisited model should also address the trade-off between privacy, accuracy, and performance. We propose a dedicated solution to address privacy requirements raised by the analysis of the ECG data for arrhythmia classification. Since the 2PC protocols cannot efficiently support all kinds of operations, we propose to design a new and customised neural network model that can be executed to classify arrhythmia accurately, and this, without disclosing the ECG data. The design of the new model customised for the use of 2PC should not result in a significant decrease on the accuracy of the classification. We remind that the goal of a neural network classification algorithm is to determine to which class a given input data belongs to.

4.4.2 PAC: Description

In order to guarantee data privacy during the arrhythmia classification phase, a dedicated neural network model should be designed. Because cryptographic techniques add a non-negligible overhead, one should optimise the complexity of the model as much as possible. Hence, the primary goal while building a new classifier is to optimise the number of neurons at each layer while keeping an adequate accuracy level.

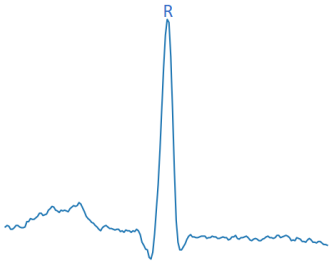


Figure 4.3 – R-peak in a heartbeat

As previously mentioned, the cryptographic technique that we choose to ensure data privacy is 2PC [92] whereby a querier holds its input, and a cloud server has the NN prediction model, namely the weight matrices and bias vectors. Similarly to [107] and [110], nonlinear NN operations such as activation functions should be replaced with more efficient operations such as low degree polynomials. In this section, we describe our privacy-by-design approach with a case study using the MIT-BIH arrhythmia dataset from the PhysioBank database[1]. We present the resulting neural network model incrementally.

We first extract heartbeats from the ECG signals. Each heartbeat is composed of 180 samples with 90 samples before the R-peak (i.e., R-peak is an upward deflection in the heartbeat as depicted in Figure 4.3), 1 sample for the R-peak, and 89 samples after the R-peak. Once heartbeats are extracted, we perform various filtering operations to create an appropriate dataset to build the neural network model. The PhysioBank database is shown in Table 4.1 and contains 23 different annotations for the extracted heartbeats.

We have decided to consider only 16 out of 23 annotations representing meaningful arrhythmia classes that have a significant number of instances in the dataset. Additionally, we realised that normal beats dominate the dataset (67.3%) and hence, result in an unbalanced dataset for model training purposes. We have reduced the number of normal beats in order for the model to predict anomalies more accurately while keeping this number sufficiently large so that it reflects reality. Moreover, we propose to use the over-sampling method to enforce the learning of low frequent classes such as class “e”. Table 4.1 provides details about the final dataset we are actually using. This dataset is further split such that 80% of the heartbeats are used to train the network, and 20% of the heartbeats are used to test the performance of the model. We propose a model with two fully connected layers involving matrix multiplications, one activation function and a final Softmax function that would provide the resulting arrhythmia class.

[1]MIT-BIH Arrhythmia Database: https://www.physionet.org/physiobank/database/mitdb/

Table 4.1 – Heartbeats for Arrhythmia classification and their frequency in our dataset

		Our Dataset		PhysioBank Dataset	
Arrhythmia Class	**Symbol**	#	%	#	%
Normal beat	N	14985	34.02%	59926	66.593%
Left bundle branch block beat	L	6450	14.64%	6450	7.168%
Right bundle branch block beat	R	5794	13.15%	5794	6.439%
Premature ventricular contraction	V	5712	12.97%	5712	6.347%
Paced beat	/	5608	12.73%	5608	6.232%
Atrial premature beat	A	2042	4.64%	2042	2.269%
Rhythm change	+	1005	2.28%	1005	1.117%
Fusion of paced and normal beat	f	786	1.78%	786	0.873%
Fusion of ventricular and normal beat	F	647	1.47%	647	0.719%
Ventricular flutter wave	!	378	0.86%	378	0.42%
Nodal (junctional) escape beat	j	184	0.42%	184	0.204%
Non-conducted P-wave (blocked APB)	x	155	0.35%	155	0.172%
Aberrated atrial premature beat	a	123	0.28%	123	0.137%
Ventricular escape beat	E	85	0.19%	85	0.094%
Nodal (junctional) premature beat	J	68	0.15%	68	0.076%
Atrial escape beat	e	26	0.06%	13	0.014%
Signal quality change	V	NA	NA	508	0.565%
Comment annotation	”	NA	NA	352	0.391%
Isolated QRS-like artifact	”	NA	NA	112	0.124%
Unclassifiable beat	Q	NA	NA	29	0.032%
Start of ventricular flutter/fibrillation	[	NA	NA	5	0.006%
End of ventricular flutter/fibrillation	]	NA	NA	5	0.006%
Premature or ectopic supraventricular beat	S	NA	NA	2	0.002%

We also optimise the number of neurons in each layer to reduce the neural network complexity:

Output layer. The number of neurons in the *output layer* corresponds to the number of arrhythmia classes. Since we only consider taking the first 16 out of 23 arrhythmia classes, the number of neurons in the output layer is set to 16.

Hidden layers. In order to choose the appropriate number of neurons within the *hidden layer* namely `FC`, we have evaluated the accuracy of models on the test dataset whereby the number of neurons varies from 2 to 100. We evaluate the overall accuracy and compute the confusion matrix that indicates the accuracy with respect to each arrhythmia class. We observe that 38 is a good choice as this implies less complexity in the model as well as its corresponding confusion matrix shows better fairness toward less frequent classes. The accuracy of our model is 96.51%. We represent the model's performance on the test dataset with the confusion matrix as illustrated in Figure 4.4.

Furthermore, in addition to the optimisation of the number of hidden neurons, we have to select the most appropriate *activation function* that cryptographic techniques (in our case 2PC) can support. Although Figure 4.5 shows better accuracy results when `ReLU` is used, we opt for the use of the `Square` function mainly for performance reasons. Indeed, the `ReLU` function involves comparison operations that can incur higher overhead compared to the `Square` function, the resulting degradation is not very significant (0.34%). Finally, we replace the Softmax function with a simple `Argmax` operation since the exponentiation

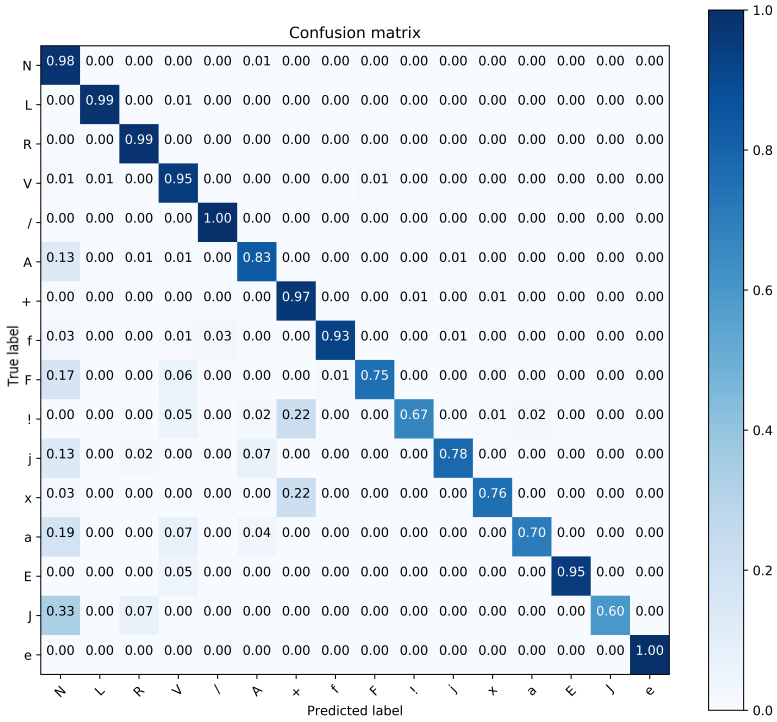


Figure 4.4 – Confusion matrix of the model for each class in the test dataset

cannot be easily computed with 2PC. Note that this approximation does not incur any accuracy loss.

Input layer. The other parameter that affects the complexity of the neural network model is the size of the *input layer*. This inherently reduces the dimension of the first matrix used for the fully connected (`FC`) layer. The main tool to adequately reduce the number of neurons of the input layer is the principal component analysis technique (PCA) [130]. PCA uses orthogonal linear transformations to find a projection of all input data (ECG heartbeats) into k dimensions which are defined as the principal components. The efficiency of using PCA for the ECG analysis domain has also been proved in [131]. It also helps reduce the noise in the ECG signals and hence improve the accuracy of the model. In order to identify the appropriate number of eigenvalues, we run a simulation with 100 hidden neurons and change the value of the input size n starting from $n = 180$ as illustrated in Table 4.2. The same simulation is executed using the the `Square` function as for the activation function. From this analysis, we choose to set the input size to 16, mainly because the resulting prediction model provides good accuracy with acceptable complexity. Hence, the number of neurons of the input layer is now set to 16.

The resulting model. To summarise, the developed model, compatible with the use of 2PC, consists of 2 `FC` layers, one activation layer implementing the `Square` (i.e., x^2) function, and one `Argmax` function. The architecture of the proposed neural network

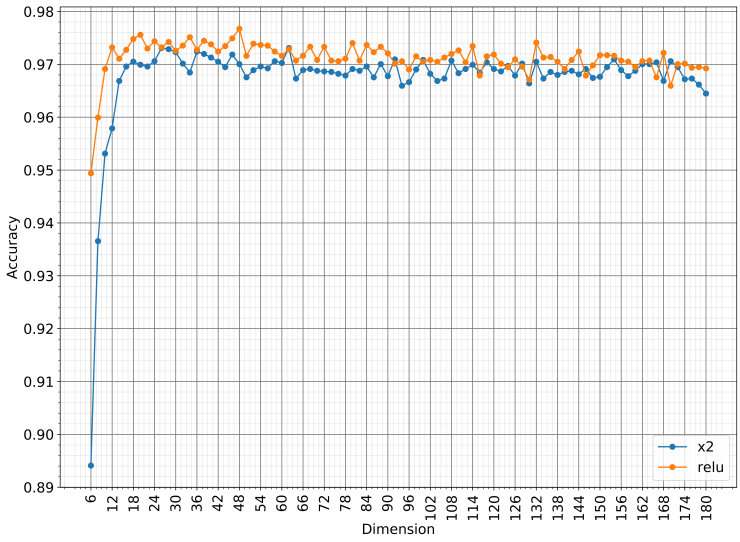


Figure 4.5 – Accuracy of the model with different dimensions of the input vector

model is shown in Figure 4.6.

The first layer consists of a `FC` layer given in Equation 4.1:

$$Y'_h = X^T.W_h + B_h. \tag{4.1}$$

where X represents the input vector (PCA transformed of a heartbeat). This input vector X of size 16 is multiplied with the weight matrix of the hidden layer, W_h, of size 16×38. This intermediate result is further added to the bias vector of the hidden layer, B_h, of size 38.

The resulting vector Y'_h becomes the input of the activation layer which consists of computing the square of each element y'_{h_i} of Y'_h described in 4.2.

$$Y_h = \begin{bmatrix} {y'_{h1}}^2 \\ {y'_{h2}}^2 \\ \vdots \\ {y'_{h38}}^2 \end{bmatrix} \tag{4.2}$$

The resulting vector Y_h is the final output of the hidden layer. This vector further becomes the input for another `FC` layer as shown in 4.3:

$$Y' = Y_h^T.W_o + B_o \tag{4.3}$$

Table 4.2 – Simulation results for the prediction model with different input sizes

Activation	Input	Training accuracy	Test accuracy
`ReLU`	180	98.10%	97.00%
`ReLU`	48	99.35%	97.32%
`ReLU`	16	98.63%	97.33%
`ReLU`	8	96.55%	96.10%
`Square`	180	97.61%	96.98%
`Square`	48	99.00%	96.70%
`Square`	16	97.68%	96.80%
`Square`	8	95.60%	95.00%

W_o, B_o Y' denote the weight matrix, the bias vector and the result of the output layer, respectively. The output is the vector Y' of size 16. Finally, the `Argmax` function is executed over the components y'_j of Y'. The result y in Figure 4.6 is the index of one of the 16 arrhythmia classes.

In total, the prediction phase consists of: $16 \times 38 + 38 \times 16 + 38 = 1254$ multiplications, $15 \times 38 + 38 + 37 \times 16 + 16 = 1216$ additions, and 1 `Argmax` operation.

4.4.3 Discussion on Principle Component Analysis

As previously mentioned, the NN model is revised and designed from scratch in order to be compatible with 2PC and remain as efficient as possible. To improve the performance of the classification phase, the size of the input is reduced using Principle Component Analysis (PCA).

PCA is a statistical method which identifies patterns, highlights similarity between elements within the dataset and finally reduces the dimension of the feature space. More formally, let D be a dataset and x_i an element of it with dimension d. The first step of PCA consists of computing the mean μ of all the elements x_i. Then, the covariance matrix A of S is computed. A has the dimension $d \times d$. The eigenvectors and corresponding eigenvalues of matrix A are further evaluated and the first k eigenvectors with the largest eigenvalues are selected. Thus, the final output of this method is a $d \times k$ matrix of the most relevant eigenvalues.

We propose to make use of the PCA method to decrease the size of the input vector. Thus, the querier would transmit less data to the cloud server when sending the input to the model. The cloud server first computes the mean μ of its dataset. Then, it computes the covariance matrix from the training dataset (for this case study, 44048 heartbeats consisting of 180 samples) and obtain the 180×16 matrix of the most relevant eigenvalues. This matrix along with the vector μ is sent to the querier who reduces the dimension of its input to 16 (instead of 180) by first normalising its signal by subtracting it with the mean vector and further multiplying the result with the received matrix. Consequently, this operation has already reduced the complexity of the proposed NN.

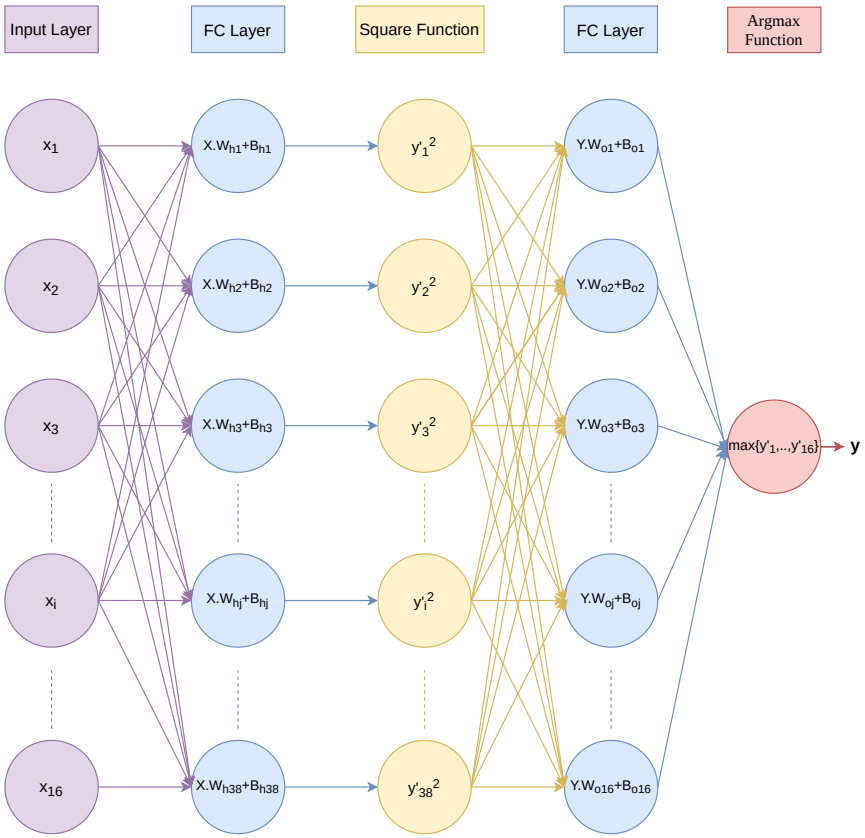


Figure 4.6 – Architecture of the neural network model in PAC

Nevertheless, the use of the PCA transformation at the querier side can result in some information leakage. We propose to analyse which information and how much information is leaked, and introduce two design approaches to avoid towards the information leakage. The leakage resulting from the use of PCA is represented by two components: the mean of the dataset and the 180×16 covariance matrix. The mean of all the signals in the training dataset does not carry any valuable information since the labels of the training signals are not included in the computation of the mean. On the other hand, the matrix of 16 eigenvectors does not correspond to the entire matrix of eigenvalues. In addition to that, without the knowledge of the eigenvalues there exist an infinite number of inverse transformations back to the original covariance matrix. Therefore, one cannot discover the training dataset and hence the model from this reduced and transformed matrix. If we choose not to leak this information while designing the privacy-preserving NN classification, then either we do not use PCA (high bandwidth and computational cost) or include the PCA steps to the 2PC solution (additional overhead but less costly). Accordingly, in PAC, we propose the following three design approaches (as shown in Figure 4.7) and evaluate the performance for each of them:

- Model 1: PCA is not integrated to 2PC (original and most efficient solution but implies some leakage),
- Model 2: PCA is integrated to 2PC (less efficient but no leakage),
- Model 3: PCA is not used (worse performance but no leakage).

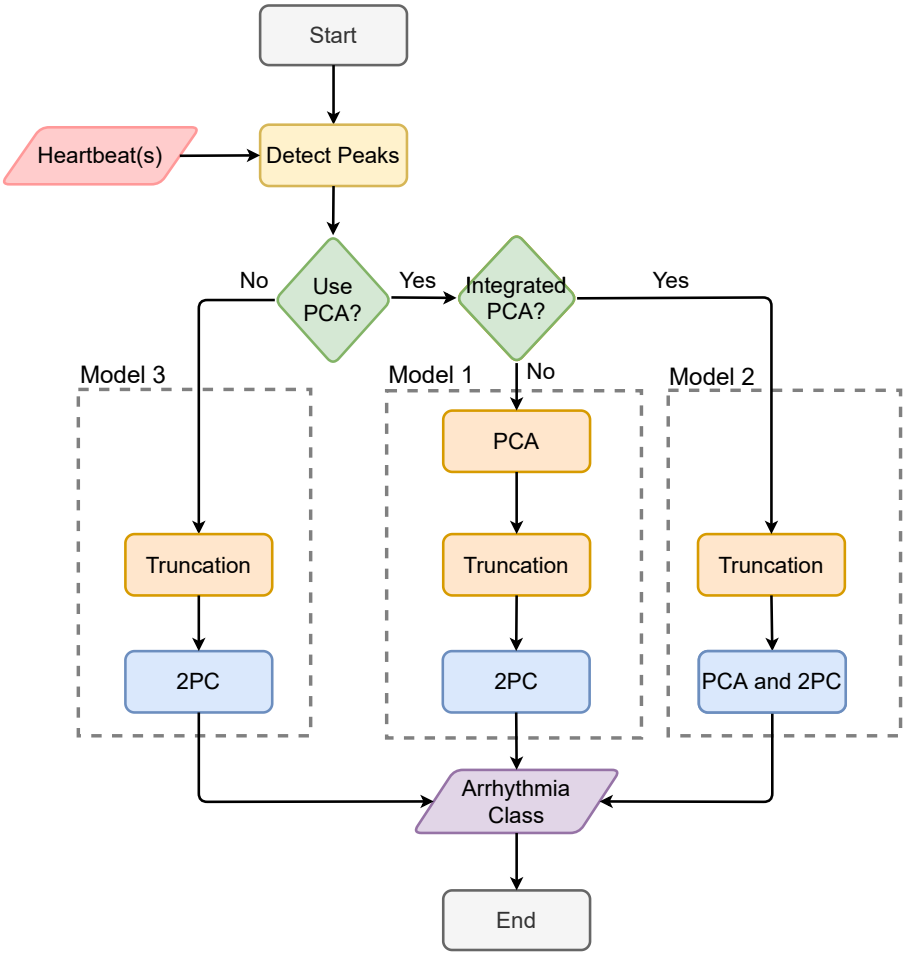


Figure 4.7 – PAC - Overview

Model 1 and Model 2 are similar: the only difference among them is on the integration of PCA into 2PC (no information leakage if Model 2 is used). Model 3 is customised to support 2PC without disclosing any information. When implementing Model 3, one can follow the similar idea of having less complex NN operations and less NN complexity that we make use of when building models 1 and 2. The resulting model has the same architecture as models 1 and 2 in terms of NN layers and the size of the output layer but the size of the input (180-sample instead of 16) and the number of the hidden neurons (40 instead of 38 neurons) are much higher. In the sequel of this section, we only describe

the implementation of Model 1. Nevertheless, the three models for PAC are implemented and performance results are provided in the next section.

4.4.4 SIMD circuits

Finally, in addition to reducing the size of the neural network and decreasing the cost of the underlying operations, we also take advantage of Single Instruction Multiple Data (SIMD) circuits which allow the packing of multiple data elements and the execution of operations in parallel. We use this technique to perform the matrix multiplications and additions more efficiently. In more details, since the number of hidden neurons is 38, the querier creates the SIMD version of its input X (of size 16) repeated 38 times (i.e. the size of the share is $38 * 16 = 608$). Similarly, the cloud server creates a SIMD version of the weight matrices W_h (of size 16x38) and W_o (of size 38x16) by flattening them to two vectors of 608 elements. Once these versions obtained, one single SIMD multiplication gate can be used to perform component-wise multiplication. Next, to finalise the matrix multiplication, some elements of the resulting vector should also be added. The cloud server also creates a SIMD version of the bias vectors and adds them to the vector resulting from the previous SIMD matrix multiplication. The `Square` activation function can also be computed using one SIMD multiplication gate. To implement the `Argmax` function, we transform the SIMD share of the previous layer to a non-SIMD share (i.e., the SIMD share is composed of 1 wire holding all the 16 values of Y' while the non-SIMD share is composed of 16 wires each wire will hold one value of Y'). Due to the inability of a comparison gates to compare between negative and positive values, we use a comparison gate to check the sign of the value by comparing it with the smallest negative number, namely -1, and then, we replace negative values with a zero using a multiplexer gate. Then, we loop on all the values (wires), and compare each of them with the max value using a comparison gate. Eventually, in each iteration two multiplexer gates are used to store the max value and max index relying on the result of the comparison gate. Hence, Equation 4.1 involves 1 SIMD multiplication and 16 SIMD additions; the activation function consists of 1 SIMD multiplication; further, Equation 4.3 is computed using 1 SIMD multiplication and 38 SIMD addition gates; finally, to evaluate the `Argmax`, 46 multiplexer gates and 31 comparison gates are used.

Moreover, we propose a secure computation of PCA in Model 2. As previously described, the computation of the PCA can eventually introduce a limited leakage of the training dataset. Therefore, a solution might be deployed by introducing the computation of the PCA vector to the 2PC model. For this to happen, the cloud server shares the mean of the dataset as well as the 16 eigenvectors using ABY (the same way it shares the weights and biases). On the other hand, the querier shares its sampled heartbeat signal. The two parties will first collaboratively compute PCA of the signal while the rest part of the circuit remains unchanged. This PCA computation layer adds 181 SIMD addition gates and 1 SIMD multiplication gate.

4.4.5 Implementation

We use the ABY framework [42] which supports all basic operations in a flexible manner using Arithmetic, Boolean or Yao's circuits to compute 2PC. ABY supports Single Instruction Multiple Data (SIMD) gates. Additionally, the ABY implementation also supports floating point representation if Boolean circuits are used. Hence, we first implement the privacy-preserving model using Boolean sharing, namely BS . Share [36]. Nevertheless, floating point representation and the use of BS . Share appear to be inefficient. We therefore propose some improvements using fixed point representations that may result in some truncations of the inputs or intermediate values in the circuit. Furthermore, since operations executed over BS . Share are much more expensive than those executed over arithmetic sharing, namely AS . Share, we propose to replace BS . Share with AS . Share as much as possible; hence, we improve PAC and set the security parameter to 128 bits in ABY.

We propose two truncation methods for the use of integers instead of real numbers:

(i) Truncation v1: The first method consists of applying truncation at intermediate stages in the circuit and hence try to continuously keep a good accuracy level. In this truncation approach, the precision of the inputs of the arithmetic circuits is in the order of the 6^{th} floating point: thus weight matrix's and input vector's values are multiplied by 2^{24} and the biases by 2^{48}. These are further converted to integers without having an impact on the precision of the value. The intermediate shares of (Y'_h) are amplified by a factor of 2^{48}. Thanks to the initial truncation, this equation does not incur any overflow. However, moving to the square function, the values need to be truncated once more. To make these truncations efficient, we propose to use a simple shift operation. Hence, shares of Y'_h are first transformed to Boolean shares and their bits are shifted by 24: This is equivalent to dividing the values by 2^{24}. To implement the shift function, the wires of position $2, 3, \ldots, 40$ (wire 1 represent the most significant bit and wire 64 represent the least significant bit) are moved to the position of the 40 most right wires (we do not move the first wire since it represent the sign bit). Then the wires of position $2, 3, \ldots, 24$ are set to the same value of the wire of position 1. This ensures correct binary representation of the values and it is compatible with negative numbers since it respects the 2's complement representation. Note that this method is also implemented in SIMD form as we perform the shift of the bits of all the values in the vector with a single operation. Once the truncation is applied, the Boolean share is re-converted to an arithmetic share and the arithmetic circuit corresponding to the activation function can be applied. Because of the multiplication operation, the resulting shares of Y_h will again be amplified by a factor of 2^{48}. The same truncation method will thus be repeated. A truncation is not needed at the fourth stage as the argmax operation only outputs the index and not the actual value. Finally, we convert Y' again to a Boolean SIMD share since comparison operations cannot be efficiently computed with arithmetic gates and implement the argmax function the same way it was implemented before. This first truncation method is evaluated on the test data and it showed good accuracy. The accuracy of the new model with the first truncation method is 96.34% which is similar to what we get from the model with the second truncation method presented next. Also, the confusion matrix of the results is

the same as the one corresponding the original model.

(ii) Truncation v2: The second method truncates the inputs before the prediction process starts, only. In this truncation approach, only the inputs to the circuits are truncated and this before the actual execution of the circuit. In order to avoid overflows, we multiply X, W_o and W_h by 10^3, B_h by 10^6 and B_o by 10^{15} and truncate the fractional part afterwards. We observe that this method is as safe as the maximum number a signed 64-bit integer variable can take is $9.223372037 \times 10^{18}$ and the upper bound for the values of Y' is 9223 and the lower bound is -9223. We observe that the risk of overflow is very low. Thanks to this approach, the actual circuit only consists of arithmetic gates except at the last stage where the `Argmax` operation needs to be executed. We have tested the accuracy of the new model using the test dataset and we have achieved an accuracy of 96.34% which is very close to the accuracy of the original model (96.51%). The confusion matrix of the new model shows the same accuracy as presented in the original model (see Figure 4.4).

Furthermore, we propose to perform arrhythmia predictions in batches, namely, with several heartbeat inputs. This can be justified as the classification of a single heartbeat may not be sufficient to diagnose the disease for a patient and the doctor may need to receive the classification of the n consecutive heartbeats.

Thanks to the use of Arithmetic circuits with Truncation v2, significant performance improvements are observed. We also realise that, with this optimisation, the online time remains very short and that 21.2% (82.2% in case of evaluation on the same machine) of the Total time corresponds to the BaseOT phase. This phase is only processed once the two parties initiate the protocol. Hence the overall time may again be decreased by performing predictions in batches (i.e. performing prediction of many ECG signals at once) using the SIMD technique, once again.

Indeed, the querier may first record N signals, prepare the inputs and further store them in a matrix $S(N)$ in Equation 4.4. Let $x_{i,j}$ be the value of the j^{th} PCA component of the signal belonging to the individual i in the dataset.

$$S(N) = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,16} \\ x_{2,1} & x_{2,2} & \dots & x_{2,16} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,16} \end{bmatrix} \tag{4.4}$$

The querier further creates the following vector $X(N)$ from $S(N)$:

$$X(N) = \underbrace{\left[\underbrace{s_1 \quad \dots \quad s_1}_{38} \quad \dots \quad \underbrace{s_N \quad \dots \quad s_N}_{38}\right]}_{16 \times 38 \times N} \tag{4.5}$$

On the other hand, the cloud server creates the weight matrix vector $W_h(N)$ (illustrated in Equation 4.6) from the original weight matrix W_h of the hidden layer.

$$W_h(N) = \left.\begin{bmatrix} w_{h1} \\ \vdots \\ w_{h38} \\ \vdots \\ w_{h1} \\ \vdots \\ w_{h38} \end{bmatrix}\right\} 16 \times 38 \times N \text{ where } W_{hi} = \begin{bmatrix} w_{h1,i} \\ w_{h2,i} \\ \vdots \\ w_{h16,i} \end{bmatrix} \tag{4.6}$$

Similarly, the output layer's SIMD weight vector $W_o(N)$ transforms the weight matrix of the output layer W_o as follows:

$$W_o(N) = \left.\begin{bmatrix} w_{o1} \\ \vdots \\ w_{o16} \\ \vdots \\ w_{o1} \\ \vdots \\ w_{o16} \end{bmatrix}\right\} 38 \times 16 \times N \text{ where } W_{oi} = \begin{bmatrix} w_{o1,i} \\ w_{o2,i} \\ \vdots \\ w_{o38,i} \end{bmatrix} \tag{4.7}$$

The cloud server also creates the vectors $B_h(N)$ and $B_o(N)$ (in Equation 4.8) from the two bias vectors B_h and B_o, respectively.

$$B_h(N) = \left.\begin{bmatrix} b_{h1} \\ \vdots \\ b_{h38} \\ \vdots \\ b_{h1} \\ \vdots \\ b_{h38} \end{bmatrix}\right\} 38 \times N,\ B_o(N) = \left.\begin{bmatrix} b_{o1} \\ \vdots \\ b_{o16} \\ \vdots \\ b_{o1} \\ \vdots \\ b_{o16} \end{bmatrix}\right\} 16 \times N \tag{4.8}$$

The Arithmetic circuit of the NN model is implemented as described in Figure 4.8 whereby only the structure of the inputs and output differ (i.e., SIMD vectors result in larger size). The number of SIMD multiplications does not change since all values are regrouped in one SIMD share and the multiplication is further performed. The number of SIMD additions also remains the same.

Finally, BS . Share which represents the `Argmax` function is also performed with SIMD gates. Values of each class in each individual output in the vector $Y(N)$ are grouped in

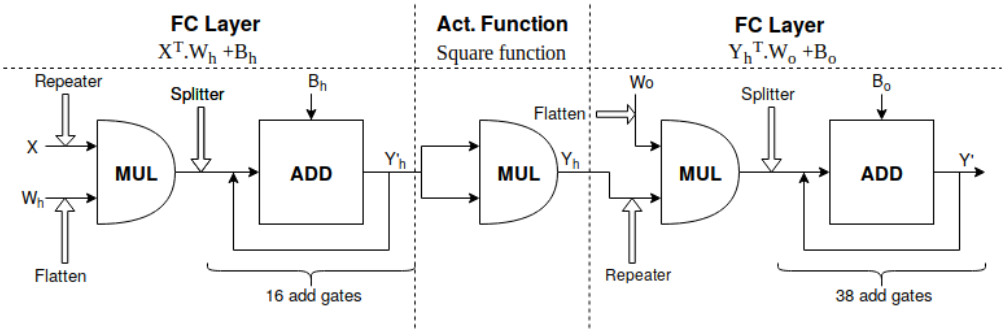


Figure 4.8 – Arithmetic circuit representation of the model with Truncation v2

separate SIMD vectors. The same comparison and multiplexers gates described before are used but this time the inputs are SIMD shares. The output of BS . Share is the vector $y(N)$ whereby each value represents the index of the class of the corresponding individual.

4.4.6 Security Evaluation

PAC performs privacy-preserving arrhythmia classifications in the presence of the semi-honest adversarial model where the querier and the cloud server have to follow the NN prediction protocol truly, yet they can try to extract information during the execution of the protocol. The goal of the querier is to hide the input data and its corresponding classification result from the cloud server. On the other hand, the cloud server does not want to reveal the NN model parameters used during computations to the querier. We employ 2PC, particularly AS . Share or BS . Share, that achieves indistinguishability given that the shares are generated from a uniformly random distribution [132]. PAC is secure and does not leak any private information about the inputs, the corresponding results, and the NN model parameters while computing the privacy-preserving arrhythmia classifications.

4.4.7 Performance Evaluation

To evaluate the computational and communication overhead of the model in a real setting, experiments were carried out by a computer which has four 3.60 GHz Intel Core i7 7700 processors, 32GB of RAM acting as the cloud server and a laptop which has two 1.70 GHz Intel Core i5 4210U processors, 4 GB of RAM acting as the querier. On the other hand, the querier and the cloud server communicate through a local area network (LAN). The querier is connected to the LAN through a wireless access point. A simulation of the bandwidth and the latency of the connection between the querier and the cloud server showed the values of 39 Mbit/sec for the bandwidth and 3.36 ms for the latency. In ABY, we set the security parameter to 128 bits.

Table 4.3 shows the performance results in terms of prediction time and bandwidth consumption for the original BS . Share as well as for both truncation approaches (namely Truncation v1 and v2) with PCA integrated and not integrated into the 2PC. We further

evaluate the model implemented by making use of the Model 2 and not use of the PCA method, Model 3. Thus, we implement Model 3 to compare with Model 2. Moreover, we have repeated all evaluations on the local set-up, i.e., the localhost on one machine, to give an insight about the overhead incured by the low bandwidth (the results are given in Table 5 in [133]). Note that the Network time in the table of performance results represents the time for the connection to be established. The Total time corresponds to the Setup time and the Online time; moreover, the Setup time corresponds to the OTExtension time and the Garbling time. Thus, the prediction time of one heartbeat is the Total time added to the baseOT time.

	Boolean Circuits	**Truncation v1**		**Truncation v2**			
Proposed neural network models	**Model 1**	**Model 1**	**Model 2**	**Model 1 without `Argmax`**	**Model 1**	**Model 2**	**Model 3**
Circuit							
Gates	553925	35477	36418	128	34329	34696	34660
Depth	4513	160	168	5	146	147	146
Time (ms)							
Total	117571.82	1218.613	2776.862	735.357	1082.804	2641.846	4723.203
Init	0.046	0.076	0.071	0.056	0.062	0.037	0.033
CircuitGen	0.046	0.074	0.062	0.067	0.078	0.055	0.047
Network	272.867	268.39	94.142	248.92	51.391	89.46	34.221
BaseOTs	288.047	309.288	310.06	311.387	291.705	294.698	298.294
Setup	107481.557	851.397	2373.818	714.511	817.807	2354.391	4409.689
OTExtension	106645.796	847.424	2369.377	714.278	816.069	2351.584	4407.521
Garbling	812.573	2.502	3.268	0.002	1.405	1.851	1.252
Online	10090.26	367.21	403.042	20.844	264.995	287.453	313.512
Data Transfer (Sent/Rcv, in KB)							
Total	319269 / 309573	2629 / 2252	7113 / 6651	1910 / 1900	2171 / 2095	6560 / 6461	12266 / 12139
BaseOTs	48 / 48	48 / 48	48 / 48	48 / 48	48 / 48	48 / 48	48 / 48
Setup	305915 / 304815	2240 / 2227	6591 / 6579	1881 / 1881	2086 / 2071	6406 / 6391	12025 / 12010
OTExtension	301095 / 304815	2057 / 2227	6377 / 6579	1881 / 1881	2053 / 2071	6373 / 6391	11992 / 12010
Garbling	4819 / 0	183 / 0	214 / 0	0 / 0	33 / 0	33 / 0	33 / 0
Online	13354 / 4757	389 / 25	522 / 72	29 / 19	85 / 24	154 / 70	240 / 129

Table 4.3 – Performance results for each model

We have also evaluated the performance of the prediction model without using any cryptographic techniques and making use of TensorFlow [102]. It takes 7.29 ms to predict one heartbeat in cleartext while this value becomes 117859 ms with PAC (without any truncation). Nevertheless, from Table 4.3, we observe some significant performance gain in terms of computational and communication cost by employing the truncation method. Compared to the model built with BS . Share, the Total time with the second truncation method is reduced by a factor of 108. As expected, the second truncation (Truncation v2) method shows some improvement over the first truncation method in terms of time and bandwidth consumption. This consumption resulted from the conversion of BS . Share and AS . Share causes overhead to the model with Truncation v1.

According to the results in Table 4.3, Model 2 still provides better results than Model 3, i.e., building the neural network model without the use of the PCA method: The time and bandwidth consumption of Model 3 is larger with a factor of 1.8 than Model 2. We have also implemented Model 2 without the `Argmax` layer (the output is a vector of 16-value where its `Argmax` can be computed locally by the querier) to show the size and performance of AS . Share without introducing any Boolean gates. Finally, the time performance presented in the table is highly affected by the low bandwidth of the communication channel between the querier and the cloud server. Moreover, we observe that the time consumption of the model evaluated locally on the same machine can reach

39.785 ms which is 27 times less than the remotely evaluated model. This limitation comes from the core of the 2PC protocol which suffers from high bandwidth consumption. We believe this problem can be easily solved by a decent connection between the querier and the cloud server.

We have run experiments with different batch sizes using the local and remote setups. The results are given in Table 4.4 for the remote setup (see Table 6 in the full paper [133] for the local setup). We can observe that the number of gates y slightly increases with respect to the number of heartbeats, which is much better than performing prediction on signals individually which will cost $y = 34329N$ gates. Also, the depth is constant regardless of the number of input heartbeats the model predicts.

Table 4.4 – Performance results for the multi-signal model

	# Input signals				
	1	**10**	**100**	**200**	**400**
Circuit					
Gates	33741	39552	40918	42422	45426
Depth	148	148	148	148	148
Time (ms)					
Total	1084.713	8002.287	77867.26	160114.6	314311
Init	0.061	0.09	0.062	0.059	0.058
CircuitGen	0.041	0.043	0.052	0.053	0.066
Network	7.115	7.681	7.513	5.34	4.307
BaseOTs	290.672	294.094	293.867	300.302	285.169
Setup	814.036	7575.32	75821.76	155921.4	306985
OTExtension	808.49	7509.797	75149.46	154673.2	304616
Garbling	5.056	62.455	650.642	1214.046	2310
Online	270.673	426.961	2045.492	4193.194	7325.77
Bandwidth (Rcv/Sent in KB)					
Total	2095 / 2167	21010 / 21652	209912 / 216247	419805 / 432465	839588 / 864898
BaseOTs	48 / 48	48 / 48	48 / 48	48 / 48	48 / 48
Setup	2071 / 2084	20782 / 20936	207647 / 209107	415276 / 418186	830533 / 836342
OTExtension	2071 / 2053	20782 / 20612	207647 / 205947	415276 / 411876	830532 / 823732
Garbling	0 / 31	0 / 315	0 / 3159	0 / 6309	0 / 12609
Online	24 / 83	227 / 716	2264 / 7139	4528 / 14278	9055 / 28555

We observe that the Total time still increases linearly with the number of signals but with a much better rate. More specifically, the batches model can decrease the time consumption with a percentage of 27% (70% in case of local evaluation) compared to performing prediction on signals one by one which takes $t = 1082.8N$ ms ($t = 39.7N$ in case of local evaluation). Finally, the BaseOT time is approximately 290 ms and remains constant regardless to the number of input signals the model predicts. This is, again, much better than performing prediction on signals, one by one, where the BaseOT time bt costs $bt = 290$ ms. Table 4.4 also shows that the bandwidth grows linear with the number of signals.

Therefore, we can conclude that prediction in batches improves the performance in terms of computational cost, but the size of the batch should be bounded according to bandwidth limitations.

4.4.8 Summary

We have presented PAC, a new Privacy-preserving Arrhythmia Classification that keeps users' ECG data confidential against service providers and the neural network model confidential against users. As a case study, we have designed a new model based on the PhysioBank dataset. The proposed model involves a customised two-layer neural network with 54 neurons. This model has been built from scratch in order to be compatible with 2PC. The solution is implemented with the ABY framework [42] which requires the truncation of input values and model parameters since it is defined over integers. The second truncation method combined with Arithmetic circuits consists of multiplying the input values with 10^3 and shows significant improvement in terms of performance and accuracy. PAC achieves an accuracy of 96.34% and experimental results show that the prediction of one heartbeat takes 1 second in real world scenarios. We show that more savings can be achieved with the use of SIMD for performing predictions in batches.

4.5 SwaNN: Switching among Cryptographic Tools for Privacy-Preserving Neural Network Predictions

In this section, we propose SwaNN [94], a privacy-preserving neural network prediction that uses an additively HE cryptosystem, namely the Paillier cryptosystem [9], and combines it with 2PC. This work was presented as a poster in two different venues: *ICT.OPEN 2019*[2] and *PUT 2019*[3] in which it was awarded as "the best poster". Moreover, SwaNN was published in *SECRYPT 2020, 17th International Conference on Security and Cryptography* [94].

4.5.1 Problem Statement

The design for a privacy-preserving NN classification solution drew the attention of researchers, and several solutions that provide privacy protection in NN predictions are proposed (See Section 4.3). The existing solutions generally rely on either (F)HE or 2PC. (F)HE-based solutions enable the computation of linear operations and low-degree polynomials of NN non-interactively, i.e., a cloud server could perform computations without any interaction with the querier. However, the solutions usually incur high computational costs due to the expensive nature of HE systems. Also, the restriction of linear and low-degree polynomial operations degrades the accuracy of prediction. 2PC-based solutions, on the other hand, provide more realistic computation performance and seem better at maintaining the accuracy of predictions. Nevertheless, the interactive nature of 2PC-based solutions leads to a higher bandwidth usage compared to HE-based alternatives. Therefore, in SwaNN, we investigate a balance between the computational and communication costs to maintain a good level of prediction accuracy.

[2]https://ict-research.nl/ict-open/
[3]https://petsymposium.org/2019/workshop.php

4.5.2 SwaNN: Description

This section presents SwaNN for privacy-preserving Convolutional Neural Network (CNN) predictions in the single-server scenario. Our main goals are, on the one hand, to reduce the computational cost of the querier and delegate this cost to the (powerful) cloud server and, on the other hand, to provide privacy for the inputs, their classification results, and the NN model.

In MLaaS, the querier has limited computation capabilities or knowledge of machine learning. Thus, it outsources the necessary computations to the cloud server who has expertise in performing machine learning with adequate computation power. We propose to provide privacy guarantees for the NN predictions using an additively HE scheme, namely Paillier [9], with 2PC. Our main goal is to minimise the computations at the querier side and the overall computational cost while maintaining privacy. In SwaNN, a querier shares a private input with a cloud server. The cloud server, which holds the CNN model, computes the prediction result on the private input. The majority of computations are performed by the cloud server, and the querier is involved when intermediate decryptions are needed. The querier encrypts an input with its public key and sends it to the cloud server, which computes the secret prediction result using the CNN parameters. Depending on the operation performed by the cloud server, the querier might be involved in the computations.

In SwaNN which is in the single-server scenario as in Figure 4.1, the majority of computations are performed by the cloud server, and the querier is involved when intermediary decryptions are needed. The querier encrypts its private input with its public key and sends the encrypted input to the cloud server, which computes the private result using the CNN model. Depending on the operation performed by the cloud server, the querier might involve in the computations.

In Section 2.1, we summarise the common layers for neural networks and the necessary operations to compute the functions in these layers. Below we explain how we can compute these layers under privacy preservation in the single-server scenario. Essentially, we separate the computations into two phases as *non-interactive phase* and *interactive phase*: (i) In the non-interactive phase, the operations are performed by the cloud server without the querier's involvement; and (ii) the interactive phase, however, requires the collaboration of the cloud server and the querier for computations. By definition, a CNN model starts with a convolutional layer. Therefore, we assume that the computations always start with an input encrypted under the Paillier cryptosystem by the querier. This encrypted input is sent to the cloud server.

Non-interactive phase

In this phase, the cloud server, who has received the encrypted input, computes the linear layers of the neural network as follows:

Convolutional Layer. The main operation in the convolutional layer is the dot product. Given an input $\mathbf{X}$ and a weight matrix $\mathbf{W}$, their dot product is computed as $\mathbf{Y} = \sum x_{i,j} \times w_{i,j}$. When the input is encrypted with the Paillier cryptosystem and the weight

matrix is in plaintext, using the homomorphic property of encryption, the dot product is computed as follows:

$$[\mathbf{Y}] = \left[\sum x_{i,j} w_{i,j}\right] = \prod [x_{i,j}]^{w_{i,j}}. \tag{4.9}$$

Since this computation does not require any decryption, it can be performed non-interactively by the cloud server.

Fully Connected Layer. Fully connected layer requires to compute a matrix multiplication. The underlying operation for matrix multiplication is the dot product, but it has to be performed for each column and row pair. Given an encrypted input as input, the fully connected layer is computed by performing Equation 4.9 repetitively.

Average Pooling Layer. Despite the computation of pooling layer is nonlinear, following the convention in the state-of-the-art works [97, 107], we use a linear approximation of the mean pooling operation. Originally, the computation of `Average` pooling requires the summation of the values within a subgroup and then a division by the subgroup size. Following the approach in [97, 107], we compute scaled `Average` pooling instead of the `Average` pooling, where the summation is performed, but the division is omitted. The scaled `Average` pooling can be computed by additive homomorphic property of the Paillier cryptosystem without interaction.

Interactive phase

In this phase, the cloud server computes the nonlinear layers of the neural network in collaboration with the querier as follows.

Activation Layer. Computing the nonlinear activation function in NN is a challenging task when privacy preservation is required. Since the Paillier homomorphic encryption scheme supports only additions, activation functions cannot be computed without performing decryption. In the existing solution on privacy-preserving neural networks, there are two approaches to compute the activation function.

The first approach is to compute a polynomial approximation of the function. CryptoNets [107] and MiniONN [97] use the `Square` (i.e., x^2) function as the approximation of the Sigmoid function. In SwaNN, we propose two solutions to compute the approximation function x^2. Since the Paillier cryptosystem does not support multiplications, as a first solution, we design an interactive secure `Square` function using the additively homomorphic property of the Paillier cryptosystem. Our solution adapts the secure multiplication protocol in [134] to a secure `Square` protocol (see Protocol 1). Our second solution for the computation of x^2 uses a multiplication operation under arithmetic sharing. The multiplication requires to switch the computations from homomorphic encryption to arithmetic sharing. Later in this section, we explain how we can perform such a switching operation.

In Protocol 1, to compute the square of the encrypted input $[x]$, the cloud server randomly chooses a number $[r]$ from $\mathbb{Z}_N$, adds it to $[x]$ to perform a secure decryption,

Protocol 1 Secure Square Protocol

Querier $(\mathsf{pk}, \mathsf{sk})$		**Cloud Server** (pk)
		$[x]$, $r \in_R \mathbb{Z}_N$
	$\xleftarrow{[x_r]}$	$[x_r] \leftarrow [x] \cdot [r]$ (this is equal to $[x+r]$)
$x_r \leftarrow \text{Decrypt}([x_r], \mathsf{sk})$		
$x_r^2 \leftarrow x_r \cdot x_r$		
$[x_r^2] \leftarrow \text{Encrypt}(x_r^2, \mathsf{pk})$	$\xrightarrow{[x_r^2]}$	
		$[x^2] \leftarrow [x_r^2] \cdot \left([r^2] \cdot [x]^{2r}\right)^{-1}$ (this is equal to $\left[x_r^2 - r^2 - 2xr\right]$)

and then sends it to the querier. The querier decrypts $[x_r]$ and takes the square of it. After square computation, the querier encrypts and sends it back to the cloud server. Then, the cloud server subtracts the underlying random values in the protocol to get the square of $[x]$.

The second approach to compute the `ReLU` activation function using 2PC. MiniONN [97] and SecureML [135] are the state-of-the-art solutions which use arithmetic circuits and Yao's Garbled Circuits [34] to compute the `ReLU` activation function. In SwaNN, we adapt a similar approach and use the circuit-based approach when the computation of `ReLU` is required. We compute `ReLU` using a comparison gate under Boolean sharing, $\text{BS}\,.\,\text{Share}$.

Max Pooling Layer. Unlike the `Average` pooling layer, we do not use an approximation function for the computation of the `Max` pooling layer. Instead, we implement the `Max` pooling using the comparison gates under $\text{BS}\,.\,\text{Share}$. We perform the `Max` pooling layer right after the activation layer to reduce the number of switching operations between 2PC and PHE.

Protocol 2 PHE to 2PC Secure Switching Protocol

Querier $(\mathsf{pk}, \mathsf{sk})$		**Cloud Server** (pk)
		$[x]$, $r \in_R \mathbb{Z}_N$
	$\xleftarrow{[x+r]}$	$[x+r] \leftarrow [x] \cdot [r]$
$x + r \leftarrow \text{Decrypt}(([x+r], \mathsf{sk})$		
$\langle x+r \rangle_q + \langle x+r \rangle_s \leftarrow \text{AS}\,.\,\text{Share}(2, x+r)$	$\xrightarrow{\langle x+r \rangle_s}$	
	$\xleftarrow{\langle r \rangle_q}$	$\langle r \rangle_q + \langle r \rangle_s \leftarrow \text{AS}\,.\,\text{Share}(2, r)$
$\langle x \rangle_q \leftarrow \langle x+r \rangle_q - \langle r \rangle_q$		$\langle x \rangle_s \leftarrow \langle x+r \rangle_s - \langle r \rangle_s$

Switching between HE and 2PC. In the previous subsections, we describe how to compute linear and nonlinear layers of neural networks using PHE and 2PC. Since linear and nonlinear operations follow each other repetitively, we need a secure switching mechanism between the two cryptographic techniques. We design a protocol for secure switching which is similar to the secure decryption mechanism described in [136]. Protocol 2 and 3

demonstrate the steps of switching from PHE to 2PC and 2PC to PHE, respectively.

Switching from PHE to 2PC (Protocol 2) requires to perform a secure decryption by masking the encrypted value with a random r. Once the querier securely decrypts the masked value $x + r$, it creates the secret shares of x for himself and for the cloud server as $\langle x + r \rangle_q$ and $\langle x + r \rangle_s$. In the mean time, the cloud server creates the secret shares of the random r as $\langle r \rangle_q$ and $\langle r \rangle_s$ to remove the mask from the original value x. Finally both parties perform a local subtraction on their shares $\langle x + r \rangle$ and $\langle r \rangle$ to compute the secret shared value $\langle x \rangle$ which is going to be used in 2PC computations.

Protocol 3 2PC to PHE Secure Switching Protocol

Querier $(\mathsf{pk}, \mathsf{sk})$		**Cloud Server** (pk)
$\langle x \rangle_q$		$\langle x \rangle_s$, $r' \in_R \mathbb{Z}_N$
	$\xleftarrow{\langle r' \rangle_q}$	$\langle r' \rangle_q + \langle r' \rangle_s \leftarrow \mathrm{AS}.\mathrm{Share}(2, r')$
$\langle x + r' \rangle_q \leftarrow \langle x \rangle_q + \langle r' \rangle_q$		
	$\xleftarrow{\langle x+r' \rangle_s}$	$\langle x + r' \rangle_s \leftarrow \langle x \rangle_s + \langle r' \rangle_s$
$x + r' \leftarrow \langle x + r' \rangle_q + \langle x + r' \rangle_s$		
$[x + r'] \leftarrow \mathrm{Encrypt}(x + r', \mathsf{pk})$	$\xrightarrow{[x+r']}$	
		$[x] \leftarrow [x + r'] \cdot [r']^{-1}$

Switching from 2PC to PHE (Protocol 3) reverses the former procedure. It starts with a secret shared value $\langle x \rangle$. Similar to the previous protocol, to prevent the leakage of the original value the parties reveal it after masking. Thus, the cloud server generates a random mask r' and sends a secret share of the random $\langle r' \rangle_q$ to the querier. Both parties perform an addition operation to mask $\langle x \rangle$, and then the cloud server sends the masked value $\langle x + r' \rangle_s$ to the querier. The querier reveals $x + r'$ by adding the two shares and encrypts it with its public key. In the final step, the cloud server removes the random mask from $[x + r']$ with a homomorphic subtraction.

4.5.3 Security Evaluation

In SwaNN, one computes neural network predictions under the privacy preservation assumption in the semi-honest adversarial model. We assume the semi-honest adversary is non-adaptive and computationally bounded. In this security model, the two parties, namely the querier and the cloud server should not be able to retrieve any additional information from the protocol execution apart from their inputs, outputs, and intermediary messages. SwaNN is secure thanks to the cryptographic techniques: Both the Paillier cryptosystem and 2PC are proven to be secure.

In the non-interactive phase of SwaNN, the security is guaranteed by the semantic security of the Paillier cryptosystem. The Paillier cryptosystem satisfies semantic security against chosen plaintext attacks under the decisional composite residuosity assumption [9]. Thus, in the computation of convolutional, fully connected, and mean pool layers, the cloud server cannot reveal any information from the encrypted messages on the condition that the encryption is performed with a key that meets the current security requirements.

The activation and `Max` pooling layers, on the other hand, requires interactive protocols between two parties during which the computations might be switched from PHE to 2PC, and vice versa. Besides the security of the Paillier cryptosystem, arithmetic secret sharing ((AS . Share) and Boolean secret sharing (BS . Share), which are used in the interactive phase of SwaNN as the 2PC building blocks, achieve indistinguishability given that the shares are generated from a uniformly random distribution [132]. Assuming that the Paillier cryptosystem and 2PC are secure, the security of the interactive phase of SwaNN can be deduced to the security of switching or decryption operations. In the rest of this section, we provide a formal security proof using the simulation paradigm [31] to show that the switching and decryption operations can be performed securely. We provide the proof only for Protocol 2, which switches the operations from PHE to 2PC.

Protocol 2 is a protocol π between a querier and a cloud server which computes the functionality f that switches the computations from PHE to 2PC. The querier does not provide an input for π (i.e. its input is an empty string $\perp$) apart from the auxiliary inputs encryption and decryption keys $(\mathsf{pk}, \mathsf{sk})$. The input of the cloud server is an ℓ-bit value x which is encrypted under the Paillier cryptosystem $[x]$. Given $[x]$, f computes $f(\perp, [x]) = \left(\langle x \rangle_q, \langle x \rangle_s\right)$ which are secret shares of x for the querier and the cloud server.

Theorem 4.5.1. *The switching protocol π (Protocol 2) securely computes the functionality $f(\perp, [x]) = \left(\langle x \rangle_q, \langle x \rangle_s\right)$ in the presence of semi-honest, non-adaptive, computationally bounded adversaries.*

Proof. In the following, we prove Theorem 4.5.1 for a corrupted cloud server and querier separately, by showing that the view of adversary $\mathcal{A}$ in the real world is computationally indistinguishable from the simulated views of $\mathcal{S}_i$, where $i \in \{c, s\}$ is for the querier and the cloud server.

- **Cloud server is corrupted by $\mathcal{A}$**: $\mathcal{S}_s$ is given the input and output of the cloud server which are $[x], \langle x \rangle_s$, and the security parameter 1^κ. In simulation, we need to show that $\mathcal{S}_s$ can generate the view of incoming messages to the cloud server, which is $\langle x + r \rangle_s$. $\mathcal{S}_s$ works as follows:

  1. $\mathcal{S}_s$ chooses a uniformly distributed random tape, r_1.
  2. $\mathcal{S}_s$ picks an $\ell + \kappa$-bit random value r' using the random tape r_1.
  3. $\mathcal{S}_s$ creates the secret shares $\langle r' \rangle_q$ and $\langle r' \rangle_s$.
  4. Using the output $\langle x \rangle_s$, $\mathcal{S}_s$ computes $\langle x + r' \rangle_s = \langle x \rangle_s + \langle r' \rangle_s$.

  The view of the cloud server in the real world is

  $$\mathbf{view}_s^\pi(\perp, [x]) = ([x], r_s; \langle x + r \rangle_s), \tag{4.10}$$

  while the view generated by the simulator

  $$\mathcal{S}_s(1^\kappa, [x], \langle x \rangle_s) = \left([x], r_1; \langle x + r' \rangle_s\right). \tag{4.11}$$

Since $\mathcal{S}_s$ does not have access to the decryption key sk, it cannot simulate Decrypt($[x + r], \mathsf{sk}$). On the other hand, it can generate the intermediary message $\langle x + r' \rangle_s$, but if r' is uniformly sampled from r_1, then

$$\{\mathcal{S}_s(1^\kappa, [x], \langle x \rangle_s), f(\perp, [x])\} \stackrel{c}{\equiv} \left\{ \mathbf{view}_s^\pi(\perp, [x]), \mathbf{output}^\pi \left(\langle x \rangle_q, \langle x \rangle_s \right) \right\}, \tag{4.12}$$

If for every nonuniform polynomial time distinguisher D there exists a negligible function $\mu(\kappa)$ such that

$$\begin{aligned} &\Big| \Pr \left[D \left(([x], r_1; \langle x + r' \rangle_s) \wedge \left(\langle x \rangle_q, \langle x \rangle_s \right) \right) = 1 \right] - \\ &\Pr \left[D \left(([x], r_s; \langle x + r \rangle_s) \wedge \left(\langle x \rangle_q, \langle x \rangle_s \right) \right) = 1 \right] \Big| \leq \mu(\kappa). \end{aligned} \tag{4.13}$$

Equation 4.13 holds due to the security of secure two-party computation and the uniformity of the random tape. The indistinguishability guarantees that a corrupted cloud server has no advantage on differentiating $\langle x + r' \rangle_s$ from $\langle x + r \rangle_s$.

- **Querier is corrupted by $\mathcal{A}$**: Different from the cloud server, the querier does not have an input for π. $\mathcal{S}_q$ is only provided the output $\langle x \rangle_q$ and the public and private keys pk, sk. To simulate the intermediary messages $[x + r]$ and $\langle r \rangle_q$, $\mathcal{S}_q$ works as follows:

  1. $\mathcal{S}_q$ chooses uniformly distributed random values r_1 and r_2.
  2. $\mathcal{S}_q$ picks an ℓ-bit random value x' and an $(\ell + \kappa)$-bit random value r' using the random tapes r_1, r_2.
  3. $\mathcal{S}_q$ calls to Encrypt $x' + r'$ as $[x' + r']$ using the public key pk.
  4. $\mathcal{S}_q$ creates secret shares for r' such that $\langle r' \rangle_q + \langle r' \rangle_s \leftarrow \mathrm{AS}.\mathrm{Share}(2, r')$.

  The view of the querier the view generated by the simulator are

$$\mathbf{view}_q^\pi(\perp, [x]) = \left(\perp, r_q; [x + r], \langle r \rangle_q \right), \tag{4.14}$$

$$\mathcal{S}_q(1^\kappa, \perp, \langle x \rangle_q) = \left(\perp, r_1, r_2; [x' + r'], \langle r' \rangle_q \right), \tag{4.15}$$

  respectively. Then,

$$\left\{ \mathcal{S}_q(1^\kappa, \perp, \langle x \rangle_q), f(\perp, [x]) \right\} \stackrel{c}{\equiv} \left\{ \mathbf{view}_q^\pi(\perp, [x]), \mathbf{output}^\pi \left(\langle x \rangle_q, \langle x \rangle_s \right) \right\} \tag{4.16}$$

  in the existence of a negligible function $\mu(\kappa)$ for every nonuniform polynomial time distinguisher D such that

$$\begin{aligned} &\Big| \Pr \left[D \left(\left(\perp, r_1, r_2; [x' + r'], \langle r' \rangle_q \right) \wedge \left(\langle x \rangle_q, \langle x \rangle_s \right) \right) = 1 \right] - \\ &\Pr \left[D \left(\left(\perp, r_q; [x + r], \langle r \rangle_q \right) \wedge \left(\langle x \rangle_q, \langle x \rangle_s \right) \right) = 1 \right] \Big| \leq \mu(\kappa). \end{aligned} \tag{4.17}$$

Equation 4.17 is correct when a semantically secure encryption scheme and secret sharing scheme are used in securing messages $[x + r]$ and $\langle r \rangle_q$ which eliminate the advantage of distinguishing $[x + r]$ from $[x' + r']$ and $\langle r \rangle_q$ from $\langle r' \rangle_q$. Using the Paillier encryption scheme, which satisfies the semantic security under the decisional composite residuosity assumption, and arithmetic secret sharing, which guarantees information theoretic security, a corrupted querier cannot break the indistinguishability. Furthermore, the adversary cannot reveal any information about x from the decryption of $[x + r]$, given that a sufficiently large, uniformly random value ($\ell + \kappa$ bits) is selected for masking x.

□

4.5.4 Performance Evaluation

We have implemented SwaNN to evaluate its performance in different settings and to compare it with state-of-the-art solutions. We used the C++ programming language for the implementation and the GMP 6.1.2 library for big integer operations. We used the ABY framework [42] for the 2PC operations. For the homomorphic operations, we used the Paillier implementation of ABY due to its efficiency. We selected 2048 bits modulus size for the Paillier operations to meet the current security standards. For the ABY operations, we selected 32-bit shares. The machine we used in the experiments runs Ubuntu 16.04 operating system with Intel Core i5-3470 CPU 3.20 GHz.

Optimising Computations

In each layer of NNs, the same operations are repeated for each index of the input independently. Thus, in our implementation, we use several optimisation techniques which help to reduce the computational and communication cost by enabling simultaneous execution. To optimise the 2PC computations, we use single instruction multiple data (SIMD) techniques [58] which are provided in ABY. SIMD techniques cannot be fully utilised for computations with the Paillier cryptosystem. Therefore, to improve the efficiency in homomorphic computations, we adapt two techniques to the Paillier encryption, which enables simultaneous computation.

The first technique we use is data packing. It packs multiple data items into a single ciphertext as described in [137]. Accordingly, we create slots of $t + \kappa$ bits for each data item where κ is the security parameter and t is the length of the data item. Given the plaintext modulus N, we can pack $\rho = \left\lfloor \frac{\log_2 N}{t+\kappa} \right\rfloor$ items in a single ciphertext as in (4.18).

$$[\hat{x}] = \sum_{m=0}^{\rho-1} [x_{i,j}] \cdot (2^{t+\kappa})^m \tag{4.18}$$

Using data packing we can use the full plaintext domain in the Paillier cryptosystem and perform additions on the packed ciphertext. Furthermore, in the interactive phase, using data packing helps reduce the bandwidth usage and the cost of decryption operations.

The second technique we use to improve efficiency is using a multi-exponentiation algorithm to simultaneously perform the operations in the form of

$$\prod_{i=1}^{w} a_i^{b_i} = a_1^{b_1} \cdot a_2^{b_2} \dots a_w^{b_w}. \tag{4.19}$$

Lim-Lee's multi-exponentiation algorithm [138,139] enables to perform (4.19) simultaneously by modifying the binary exponentiation algorithm using several pre-computation techniques. In our work, we can apply multi-exponentiation for the computation of dot product (in Equation 4.9) over encrypted data thanks to the additive homomorphism of the Paillier cryptosystem.

We summarise the optimisations used in each layer of (C)NN as follows:

- The `Conv` Layer: Multi-exponentiation technique is used to minimise the cost of dot products.
- The `Act` Layer: Data packing is used before performing the activation function. If activation is performed with 2PC operations, then SIMD optimisation is used.
- The `Pool` Layer: No optimisation technique is needed.
- The `FC` Layer: Multi-exponentiation technique is used to reduce the cost of matrix multiplications.

Experiments

We run three experiments with respect to the activation function used in NNs. In the first experiment, we used x^2 as the activation function and re-trained the neural network structure in CryptoNets [107]. In the second, we used `ReLU` as the activation function and respectively re-trained the neural network structure for MNIST and Cifar-10 in MiniONN [97]. In the final experiment, we use the same NN structure for the ECG dataset in [93] to show the performance results of SwaNN. The properties of the neural networks are detailed in Section 4.4.

Neural Network Structures. In our experiments, we use several neural network structures of which two are previously trained by CryptoNets [107] and MiniONN [97] to perform input classification on the MNIST data. Table 1 in CryptoNets [107] summarises the structure of the proposed convolutional neural network (CNN). This network we employ in our experiment contains 2 `Conv`, 2 activation of x^2, 2 Scaled Mean pooling, and 2 `FC` layers and achieves 98.95% of accuracy. Secondly, we employ the neural network structure proposed in MiniONN (Figure 12 in [97]) for MNIST. The accuracy of the network is 99.31%. The activation function of the network is `ReLU`. `Max` pooling is used in the pooling layer. Finally, we utilise the NN structure trained by [93] to perform the ECG signal classification. Table 4.6 represents the structure of the neural network consisting of 2 `FC` layers and 1 x^2 activation. This network achieves accuracy of 96.34%.

Experiment 1. In the first experiment, we benchmark the performance of SwaNN with x^2 as an activation function as given in Table 4.5. We design two different cryptographic techniques based settings: (i) The first setting is an *only-PHE* setting which is totally based on the Paillier cryptosystem and implements x^2 as described in Protocol 1; and (ii) The second setting is a *hybrid setting* where the CNN predictions computation switches between PHE and 2PC and which implements the secure switching protocols in Protocols 2 and 3 with the activation function x^2 implemented using the ABY framework.

Table 4.5 – Computation time per layer (in ms). The timings are provided for optimised and non-optimised PHE-only setting and optimised hybrid setting.

	Non-optimised - PHE only		Optimised - PHE only		Optimised - Hybrid	
Layer	Querier	Cloud Server	Querier	Cloud Server	Querier	Cloud Server
`Conv`	–	1831	–	892	–	917
`Act`	12651	15805	2487	19253	2292	566
`Pool`	–	35	–	34	–	33
`Conv`	–	2799	–	1329	–	1386
`Pool`	–	37	–	38	–	37
`FC`	–	6420	–	3809	–	3989
`Act`	1504	1879	314	2231	273	266
`FC`	–	10	–	10	–	11
Total	42972		30399		9841	

The results show that when no optimisations are used, the prediction of one input is computed approximately in 43 seconds. However, when we use optimisation techniques, we can reduce the computation time to 30 seconds. SwaNN in the hybrid setting reduces this cost to 10 seconds (i.e., 75% less).

In Table 4.6, we show the details of the computation time for the activation layer in the hybrid setting. The packing, decryption and unpacking operations are performed during the switching from PHE to 2PC. The encryption algorithms are computed by both parties when switching the operations from 2PC to PHE. While the querier takes 2.3 seconds for the computations, the cloud server takes approximately 581 milliseconds.

Table 4.6 – Computation time for the activation layer for the hybrid setting (in ms).

Operation	Querier	Cloud Server
Packing	–	409
Decryption	72	–
Unpacking	0.1	–
ABY	11	14
Encryption	2220	158
Total		2884

As a final analysis, in Table 4.7 we compare SwaNN with the state-of-the-art works CryptoNets [107] and MiniONN [97] with respect to computation time and bandwidth

usage. The performance results of CryptoNets and MiniONN are taken from the respective papers. CryptoNets employing FHE evaluates 297.5 seconds for one prediction. The protocol enables simultaneous computation by packing 4096 inputs into a single ciphertext. This is an advantage when the same querier has a very large number of prediction requests. MiniONN can compute the prediction result for the same CNN model in 1.28 seconds. However, this computation requires 47.6 MB bandwidth usage, whereas SwaNN enables the same prediction result in 9.8 seconds with 1.69 MB bandwidth use.

Table 4.7 – Comparison with the state-of-the-art in Exp. 1.

	Computation time (s)	Bandwidth usage (MB)
CryptoNets [107]	297.5	372.2
MiniONN [97]	1.28	47.6
SwaNN	9.8	1.69

Experiment 2. As the second experiment, we benchmark the performance of SwaNN with `ReLU` activation function for the CNN structure as in [97]. We employ the maximum operation for pooling layers. We provide the timings for the `Max` pooling along with `ReLU` function since we implemented them together. We measure the timings only with optimisations. Table 4.8 details the computation time for each layer.

Table 4.8 – Computation time per layer (in ms).

Layer	Querier	Cloud Server
`Conv`	–	10192
`Act+Pool`	6852	2593
`Conv`	–	1148
`Act+Pool`	778	467
`FC`	–	1325
`Act`	274	508
`FC`	–	5
Total	24242	

Due to larger number of input size in each NN layer, the computation cost of SwaNN reaches to 24 seconds for one input classification. The first activation layer is the dominant layer in the run time. As expected, the high computation cost is caused by the decryption operations which are performed during the switching phase from PHE to 2PC.

In Table 4.9, we compare the performance of SwaNN with MiniONN. Clearly, MiniONN outperforms SwaNN almost 3-fold in computation time. However, in terms of communication, SwaNN is more efficient with a bandwidth usage of 160 MB (compared to 657 MB in MiniONN).

Experiment 3. As the third experiment, we measure the performance of SwaNN with the NN architecture containing the `Square` activation function (namely, Model 1) described in PAC [93].

Table 4.9 – Comparison with the state-of-the-art in Exp. 2.

	Computation time (s)	Bandwidth usage (MB)
MiniONN [97]	9.32	657.5
SwaNN	24.00	160.9

In the third experiment, the querier perform some pre-computation: Multiplying each entry of the ECG data with others, including itself, only once. Further, the querier encrypts them and its ECG data and sends them to the cloud server. When receiving the encrypted data, the cloud server computes the NN operations over them, obtains the encrypted result, and sends it to the querier. Therefore, there is no interaction between the querier and the cloud server for computing the `Square` layer. Therefore, we call this setting *Swann w/o interaction*.

Table 4.10 – Comparison with the state-of-the-art in Exp. 3.

	Computation time (s)	Bandwidth usage (MB)
PAC (see Section 4.4)	1.37	4.36
SwaNN	8.19	0.03
SwaNN w/o interaction	1.51	-

In Table 4.10, we compare the performance of SwaNN with PAC [93]. According to results from Table 4.10, PAC outperforms SwaNN almost 6-fold in computation time. However, in terms of communication, SwaNN is more efficient with a bandwidth usage of 0.03 MB (compared to 4.36 MB in PAC). Moreover, SwaNN w/o interaction is more efficient than PAC regarding bandwidth usage and timing cost for SwaNN is close to PAC.

4.5.5 Summary

We have designed a privacy-preserving neural network prediction protocol that combines the additively homomorphic Paillier encryption scheme with 2PC. Thanks to the use of the Paillier encryption algorithm for linear operations and also the x^2 activation function, the solution achieves better computational cost compared to existing HE-based solutions. Different computation optimisations based on the use of data packing and the multi-exponentiation algorithm have been implemented. Furthermore, the communication cost is also minimised since 2PC is only used for nonlinear operations (`Max` and/or `ReLU`). Experimental results show that SwaNN actually achieves the best of both worlds, namely, better computational overhead compared to FHE-based solutions and, better communication overhead compared to 2PC-based solutions.

4.6 ProteiNN: Privacy-preserving one-to-many Neural Network classification

In this section, we propose ProteiNN [95], a privacy-preserving neural network prediction when the NN model is in the encrypted form in the cloud. This work was published in *SECRYPT 2020, 17th International Conference on Security and Cryptography.*

4.6.1 Problem Statement

The problem of privacy-preserving neural networks (NN) classification has already been studied by many researchers (see Section 4.3 for the state-of-the-art). Most of these works consider that the party who performs the NN operations is the model provider and is sufficiently powerful; therefore, the NN model is kept in plaintext form. On the other hand, [114] and [115] allow the outsourcing of these operations to the cloud server but the only querier, in this case, is the model provider. In both cases, the goal is to perform NN operations over queriers' data without leaking any information, including the model. ProteiNN proposes to extend this scenario that we name the *one-to-one scenario*, by enabling *one* model provider to securely outsource its model to a cloud server and consider a *one-to-many scenario* whereby different queriers can query the model. Additionally, while delegating the NN operations to the cloud server, the model provider also wishes to maintain the control over the use of this model by legitimate and authorised queriers only.

To further illustrate the importance of the one-to-many setting as illustrated in Figure 4.2, we define a scenario whereby one party, such as a healthcare analytics company, owns a NN model $\mathbf{M}$ to classify a particular disease. This company can later use $\mathbf{M}$ to decide whether a particular patient suffers from this disease or not. Moreover, this company wants to make $\mathbf{M}$ profitable to many of its customers (such as hospitals or doctors) who are willing to diagnose the disease over their input denoted $\mathbf{X}$ using $\mathbf{M}$. With this aim, $\mathbf{M}$ is outsourced to the cloud server. Before outsourcing $\mathbf{M}$, the healthcare analytics company needs to encrypt $\mathbf{M}$ to protect its intellectual property. Later on, customers who want to query $\mathbf{M}$ send their encrypted input $\mathbf{X}$ to the cloud server. The cloud server basically applies encrypted $\mathbf{M}$ over encrypted $\mathbf{X}$ originating from authorised customers.

4.6.2 Threat Model

Our threat model differs from the previous ones since the NN model is unknown to CS and we also consider potential collusion attacks. More specifically, we assume that all potential adversaries are semi-honest, i.e., parties adhere to the protocol steps but try to obtain some information about the model, the input or the result. Moreover, given the one-to-many setting and the introduction of the additional cloud server, we assume that collusions between CS and Q_i, between CS and MP, and between Q_i and MP may exist.

In this threat model, queriers aim at keeping their input $\mathbf{X}$ and the corresponding result $\mathbf{Y}$ secret from CS and MP. On the other hand, MP does not want to disclose $\mathbf{M}$ to

CS and Q_i. To summarise, ProteiNN considers the following potential adversaries: (i) An external adversary (who does not participate in ProteiNN) who should not learn any information about model $\mathbf{M}$, input $\mathbf{X}$, and result $\mathbf{Y}$; (ii) Q_i who should not learn any information about model $\mathbf{M}$ even if Q_i and CS collude. Note that similar to previous works, we omit the attacks whereby Q_i can try to re-build the model based on the authorised results that it receives.; (iii) CS who should not discover model $\mathbf{M}$ (Apart from its architecture), input $\mathbf{X}$, and the corresponding result $\mathbf{Y}$ even if CS colludes with MP or querier(s); (iv) MP who should not learn anything about input $\mathbf{X}$ and its result $\mathbf{Y}$ even when it colludes with CS or querier(s).

Based on this threat model, we define the following privacy requirements: (i) Model $\mathbf{M}$ is unknown to all parties in the protocol except MP. This requirement is usually not addressed by state-of-the-art solutions. (ii) Input $\mathbf{X}$ and result $\mathbf{Y}$ are only known by the actual querier Q_i and this, only if authorised by MP.

4.6.3 ProteiNN: Description

We consider the following two main problems arose by the one-to-many scenario: (i) each party uses a different public key to encrypt its data (model for MP and inputs for Q_i) and (ii) queries received from Q_i should only be processed if MP authorises them. This setting implies that both the model and the queries should be encrypted with the same key at the classification step. With this aim, we introduce a Trusted Third Party (TTP) and use its public key as the common encryption key for both the model and the queries. TTP is considered as being offline: It does not play any role during the classification phase; it only distributes keying materials. H-PRE is used towards the end of the classification phase, i.e., when Q_i needs to decrypt the actual result: Indeed, the result encrypted with TTP's public key needs to be re-encrypted with Q_i's public key.

A preliminary setup phase where each party reaches TTP in order to receive their relevant keying material is first defined. TTP is considered offline during the subsequent classification phase.

Setup Phase

During the setup phase depicted in Figure 4.9, all the relevant keying material is distributed and the encrypted model is sent to the cloud server. Namely, each querier Q_i generates a pair of public-secret keys; TTP generates a pair of public-secret keys and a set of re-encryption keys allowing re-encryption from TTP's public key to queriers' public key (one for each querier). The reason why TTP generates re-encryption keys is to enable MP to authorise a given classification request. Once public keys and re-encryption keys are received, MP encrypts its model with the public key of TTP and sends it to CS.

In more details, the setup phase consists of the following steps:

1. For given public parameters pp,
   TTP: $(\mathsf{pk}_{\mathsf{TTP}}, \mathsf{sk}_{\mathsf{TTP}}) \leftarrow \text{H-PRE}.\text{KeyGen}(\mathsf{pp})$
   Q_i: $(\mathsf{pk}_{\mathsf{Q}_i}, \mathsf{sk}_{\mathsf{Q}_i}) \leftarrow \text{H-PRE}.\text{KeyGen}(\mathsf{pp})$ where $1 \leq i \leq n$.
2. TTP: $\mathsf{rek}_{\mathsf{TTP}\rightarrow\mathsf{Q}_i} \leftarrow \text{H-PRE}.\text{ReKeyGen}(\mathsf{sk}_{\mathsf{TTP}}, \mathsf{pk}_{\mathsf{Q}_i})$, $\forall\ 1 \leq i \leq n$.

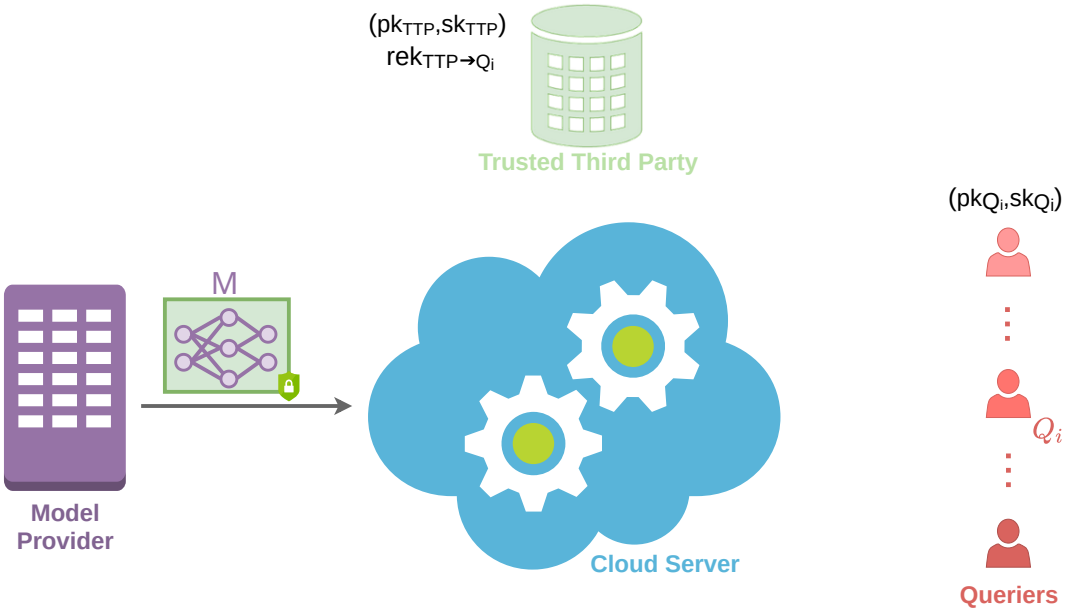


Figure 4.9 – ProteiNN - Setup phase

3. MP: $[\mathbf{M}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Encrypt}(\mathbf{M}, \mathsf{pk}_{\mathsf{TTP}})$
   Send $[\mathbf{M}]_{\mathsf{TTP}}$ to CS.
4. MP: Generate random vectors $\mathsf{r}_{ij}, \mathsf{r}'_{ij} \in_R \mathbb{Z}_N$ for Q_i where $1 \leq j \leq l$ is the query number.
5. MP: $[\mathsf{r}_{ij}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Encrypt}(\mathsf{r}_{ij}, \mathsf{pk}_{\mathsf{TTP}})$
   $[\mathsf{r}'_{ij}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Encrypt}(\mathsf{r}'_{ij}, \mathsf{pk}_{\mathsf{TTP}})$
   Store them locally.
   Send them to Q_i.

Classification Phase

The classification phase of ProteiNN that is described below is illustrated in Figure 4.10. As previously mentioned, TTP is not involved in this phase.

1. Q_i: $[\mathbf{X}_{ij}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Encrypt}(\mathbf{X}_{ij}, \mathsf{pk}_{\mathsf{TTP}})$.
   Randomise $[\mathbf{X}_{ij}]_{\mathsf{TTP}}$ using $\text{H-PRE}.\text{Eval}(+, ([\mathbf{X}_{ij}]_{\mathsf{TTP}}, [\mathsf{r}_{ij}]_{\mathsf{TTP}}))$.
   Randomise $[\mathbf{X}_{ij}]_{\mathsf{TTP}}$ using $\text{H-PRE}.\text{Eval}(\cdot, ([\mathbf{X}_{ij}]_{\mathsf{TTP}}, [\mathsf{r}'_{ij}]_{\mathsf{TTP}}))$.
   Send $[\mathbf{X}_{ij} + \mathsf{r}_{ij}]_{\mathsf{TTP}}$ and $[\mathbf{X}_{ij} \cdot \mathsf{r}'_{ij}]_{\mathsf{TTP}}$ to CS.
2. CS: Randomise $[\mathbf{X}_{ij} + \mathsf{r}_{ij}]_{\mathsf{TTP}}$ using $\text{H-PRE}.\text{Eval}(+, ([\mathbf{X}_{ij} + \mathsf{r}_{ij}]_{\mathsf{TTP}}, [\mathsf{s}_{ij}]_{\mathsf{TTP}}))$ where $\mathsf{s}_{ij} \in_R \mathbb{Z}_N$.
   Forward $[\mathbf{X}_{ij} + \mathsf{r}_{ij} + \mathsf{s}_{ij}]_{\mathsf{TTP}}$ to MP with the identifier of the querier (id_i).
3. MP: Perform the following homomorphic operations over the received query and send the outcome to CS if Q_i is authorised.

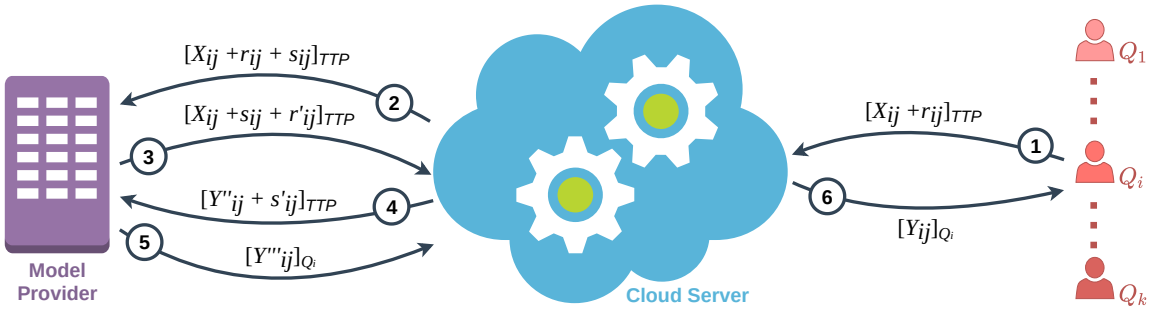


Figure 4.10 – ProteiNN - Classification phase

(a) $[\mathbf{X}_{ij} + \mathsf{s}_{ij}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Eval}(-, ([\mathbf{X}_{ij} + \mathsf{r}_{ij} + \mathsf{s}_{ij}]_{\mathsf{TTP}}, [\mathsf{r}_{ij}]_{\mathsf{TTP}}))$

(b) $[\mathbf{X}_{ij} + \mathsf{s}_{ij} + \mathsf{r}'_{ij}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Eval}(+, ([\mathbf{X}_{ij} + \mathsf{s}_{ij}]_{\mathsf{TTP}}, [\mathsf{r}'_{ij}]_{\mathsf{TTP}}))$

Note that if/whenever MP does not want to authorise querier Q_i, MP can send a reject message to CS, and thus, CS would terminate the protocol for Q_i.

4. CS: $[\mathbf{X}_{ij} + \mathsf{r}'_{ij}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Eval}(-, ([\mathbf{X}_{ij} + \mathsf{s}_{ij} + \mathsf{r}'_{ij}]_{\mathsf{TTP}}, [\mathsf{s}_{ij}]_{\mathsf{TTP}}))$
Perform the classification, i.e.,

$$[\mathbf{Y}'_{ij}]_{\mathsf{TTP}} = [\mathbf{M}(\mathbf{X}_{ij} + \mathsf{r}'_{ij})]_{\mathsf{TTP}} = \text{H-PRE}.\text{Eval}(\mathbf{C}, ([\mathbf{M}]_{\mathsf{TTP}}, [\mathbf{X}_{ij} + \mathsf{r}_{ij}]_{\mathsf{TTP}}))$$

where $\mathbf{C}$ is the circuit containing the NN layers' linear operations.
$[\mathbf{Z}_{ij}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Eval}(\mathbf{C}', ([\mathbf{M}]_{\mathsf{TTP}}, [\mathbf{X}_{ij} \cdot \mathsf{r}'_{ij}]_{\mathsf{TTP}}))$ where where $\mathbf{C}'$ is a circuit containing some layers' operations.
$[\mathbf{Y}''_{ij}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Eval}(-, ([\mathbf{Y}'_{ij}]_{\mathsf{TTP}}, [\mathbf{Z}_{ij}]_{\mathsf{TTP}}))$
Randomise $[\mathbf{Y}''_{ij}]_{\mathsf{TTP}}$ with $[\mathsf{s}'_{ij}]_{\mathsf{TTP}}$ using $\text{H-PRE}.\text{Eval}(+, (.))$ where $\mathsf{s}'_{ij} \in_R \mathbb{Z}_N$.
Send them to MP.

In case some revocation of Q_i occurs, MP has, once again, the opportunity to reject the query and will not do any re-encryption.

5. MP: $[\mathbf{Z}'_{ij}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Eval}(\mathbf{C}'', ([\mathbf{M}]_{\mathsf{TTP}}, [\mathbf{X}_{ij} \cdot \mathsf{r}'_{ij} + \mathsf{s}''_{ij}]_{\mathsf{TTP}}))$ where $\mathbf{C}''$ is a circuit containing some layers' operations to compute the correct result $\mathbf{Y}_{ij}$ requested by Q_i.
$[\mathbf{Y}'''_{ij}]_{\mathsf{TTP}} \leftarrow \text{H-PRE}.\text{Eval}(-, ([\mathbf{Y}''_{ij} + \mathsf{s}'_{ij}]_{\mathsf{TTP}}, [\mathbf{Z}'_{ij}]_{\mathsf{TTP}}))$
$[\mathbf{Y}'''_{ij}]_{\mathsf{Q}_i} \leftarrow \text{H-PRE}.\text{ReEncrypt}(\mathsf{rek}_{\mathsf{TTP}\rightarrow\mathsf{Q}_i}, [\mathbf{Y}'''_{ij}]_{\mathsf{TTP}})$
Send $[\mathbf{Y}'''_{ij}]_{\mathsf{Q}_i}$ to CS.

6. CS: $[\mathbf{Y}_{ij}]_{\mathsf{Q}_i} \leftarrow \text{H-PRE}.\text{Eval}(-, ([\mathbf{Y}'''_{ij}]_{\mathsf{Q}_i}, \mathsf{s}'_{ij}]_{\mathsf{TTP}}))$
Send $[\mathbf{Y}_{ij}]_{\mathsf{Q}_i}$ to Q_i.

7. Q_i: $\mathbf{Y}_{ij} \leftarrow \text{H-PRE}.\text{Decrypt}(\mathsf{sk}_{\mathsf{Q}_i}, [\mathbf{Y}_{ij}]_{\mathsf{Q}_i})$.

4.6.4 Security Evaluation

We analyse the security of ProteiNN considering our newly introduced threat model and show that it satisfies the privacy requirements defined in Section 4.6.1. We propose to conduct this security analysis incrementally by taking each adversary into account, one by one, and the potential collusions. As described in Section 4.6.1, all ProteiNN parties are considered as semi-honest adversaries. Furthermore, we assume that the H-PRE scheme that ProteiNN uses is semantically secure and that the encrypted addition of random vectors r_{ij}, r'_{ij}, s_{ij}, and s'_{ij} is considered as perfectly secure.

Privacy against external adversaries. During the classification phase, all parties encrypt their input, result, or model using H-PRE. Hence, an external adversary can only obtain encrypted information exchanged among ProteiNN players. Given the semantical security of H-PRE and the perfect secrecy of the simple additive encryption scheme, an external adversary who does not participate in ProteiNN and who does not hold any keying material cannot learn any information about $\mathbf{M}$, $\mathbf{X}$, and $\mathbf{Y}$.

Privacy against adversary Q_i***.*** The goal is to achieve model privacy against Q_i. In ProteiNN, $\mathbf{M}$ is encrypted by with $\mathsf{pk}_{\mathsf{TTP}}$. Assuming that the underlying H-PRE is semantically secure and since Q_i does not know $\mathsf{sk}_{\mathsf{TTP}}$, Q_i cannot recover $\mathbf{M}$ in plaintext. Furthermore, Q_i can also try to learn the input of another querier Q_t and the corresponding result. In this case, Q_i becomes an external adversary as it does not have any role in the protocol executed between CS and Q_t. Hence, ProteiNN is also secure in this case. Finally, even if multiple queriers collude, they do not succeed in any leakage of the model.

Privacy against adversary CS***.*** All the information that CS receives are encrypted with $\mathsf{pk}_{\mathsf{TTP}}$ or $\mathsf{pk}_{\mathsf{Q}_i}$. Thanks to the security of underlying building blocks, a semi-honest CS cannot discover $\mathbf{M}$ (apart from its architecture), $\mathbf{X}$, and $\mathbf{Y}$.

Privacy against adversary MP***.*** A semi-honest MP can try to discover queriers' inputs and the corresponding classification results. Since these randomised information are encrypted with $\mathsf{pk}_{\mathsf{TTP}}$ and $\mathsf{pk}_{\mathsf{Q}_i}$, respectively, and since MP does not hold the corresponding secret keys, MP cannot learn these inputs and results.

MP-CS ***collusions.*** We have already shown that ProteiNN is secure against MP and CS, individually. The collusion of these two players do not help them discover additional information since all inputs and results are encrypted using the semantically secure H-PRE with $\mathsf{pk}_{\mathsf{TTP}}$ and $\mathsf{pk}_{\mathsf{Q}_i}$, and results are re-encrypted with $\mathsf{rek}_{\mathsf{TTP}\to\mathsf{Q}_i}$.

Q_i-$\mathsf{CS} ***collusions.*** Collusions between Q_i and CS do not result in any leakage regarding $\mathbf{M}$, other queriers' inputs, and results. Indeed, thanks to the use of random vectors r_{ij} and r'_{ij} at the classification phase, even if a malicious Q_i shares its keying material with CS to discover $\mathbf{X}$ from another legitimate Q_t, both adversaries cannot retrieve it because of its randomisation with $r_{\mathsf{Q}_{tj}}$.

Q_i-$\mathsf{MP} ***collusions.*** Collusions between Q_i and MP do not result in any leakage regarding other queriers' inputs and results thanks to the randomisation of both input and result.

More precisely, even if malicious Q_i and MP collude to discover $\mathbf{X}$ and $\mathbf{Y}$ of legitimate Q_t, they cannot retrieve them because of the use of random vectors s_{tj} and s'_{tj}.

Note on Q_i-MP-CS collusions. We consider that all three players cannot collude since in this case there is no need for privacy protection.

Note on Multiple MP case. ProteiNN can easily be extended to a *many-to-many scenario* involving many model providers using multiple instances of ProteiNN for each model provider and its queriers.

4.6.5 Performance Evaluation

We evaluate the performance of ProteiNN using the arrhythmia detection case study described in Section 4.4 whereby MP owns a NN model $\mathbf{M}$ for the classification of heart arrhythmia; Queriers' inputs consist of the individuals' Electro-Cardiogram (ECG) data and the result is the actual arrhythmia type the patient suffers from. The underlying NN model, namely Model 1 consists of two `FC` layers and one activation layer implemented with the `Square` function: (i) It involves 16 input neurons, 38 hidden neurons, and 16 output neurons, and (ii) The model provides an accuracy of 96.34% (see Section 4.4.5).

Table 4.11 – Performance results for ProteiNN

Setup phase		
ProteiNN Step	**Player**	**Time (ms)**
PRE.KeyGen	TTP	20.02
PRE.KeyGen	Q_i	21.18
PRE.ReKeyGen	TTP	267.57
PRE.Encrypt	MP	1514.08
Random number generation & PRE.Encrypt	MP	55.69
Classification phase of one input		
ProteiNN Step	**Player**	**Time (ms)**
PRE.Encrypt	Q_i	31.74
PRE.Eval for random numbers	Q_i	80.18
PRE.Eval for input randomisation & random generation	CS	27.14
PRE.Eval for random removal	MP	1.95
PRE.Eval for random addition	MP	1.93
PRE.Eval for random removal	CS	1.77
PRE.Eval for classification	CS	26898.37
PRE.Eval for random removal	CS	59.49
PRE.Eval for result randomisation & random generation	CS	31.78
PRE.Eval for random removal	MP	69.26
PRE.ReEncrypt	MP	30.12
PRE.Eval for random removal	CS	1.84
PRE.Decrypt	Q_i	5.57
TOTAL		27209.40

Experimental setup

To implement ProteiNN, we have utilised the PALISADE library (v1.5.0) supports several HE schemes and their PRE versions. The H-PRE scheme we employ for ProteiNN is

H-BFVrns [140] mainly because it is the most efficient in PALISADE. We follow the standard HE security recommendations (e.g., 128-bit security) indicated in [141] for H-BFVrns. The ProteiNN steps for TTP and CS were carried out using a desktop computer with 4.0 GHz Intel Core i7-7800X processor, 128 GB RAM, and the Ubuntu 18.04.3 LTS operating system whereas the steps for Q_i and MP were performed using a laptop with 1.8 GHz Intel Core i7-8550U processor, 32 GB RAM, and the Ubuntu 18.10 operating system.

We have evaluated the performance of both the setup and classification phases. Detailed results are depicted in Table 4.11. These results correspond to the average from the execution of 100 individual simulations. We observe that one ProteiNN classification instance takes 27.2 s, approximately. Only the cloud server performs costly operations. Indeed, the querier takes around 31 and 5 ms, to encrypt and decrypt its input and result, respectively. The cost of ReEncrypt (about 30 ms) seems negligible when compared to the cost of the classification phase. Among the operations performed by MP, the most costly one is the encryption of $\mathbf{M}$ (about 1.5 s). It is worth to note that this operation is performed during the setup phase and only once.

Table 4.12 – Performance results for ProteiNN players

Setup phase	
ProteiNN Player	**Time (ms)**
TTP	287.59
Q_i	21.18
MP	1569.77
Classification phase	
ProteiNN Player	**Time (ms)**
Q_i	85.75
MP	103.26
CS	27020.39

As shown in Table 4.12, we observe that while CS takes 27 s to classify a heartbeat, MP and Q_i only take 103 ms and 86 ms, respectively. We have also evaluated the classification cost for MP in a one-to-one scenario in order to justify the need for the cloud servers.

To summarise, our study shows that outsourcing machine learning operations in a privacy-preserving manner, in a one-to-many scenario, is possible and that relieves the computation burden from the model provider to the cloud server which is assumed more powerful.

4.6.6 Summary

We have proposed ProteiNN, a privacy-preserving one-to-many NN classification solution that is based on the use of H-PRE and a simple additive encryption. ProteiNN achieves confidentiality for the model(s), the inputs, and the results. Additionally, the model provider also has control over the model outsourced the cloud server. We have provided

a detailed security analysis by considering all potential adversaries including collusions among them. We have implemented ProteiNN as a proof-of-concept with a case study and our work shows promising performance results and calls for future work to evaluate the scalability of ProteiNN. We believe that with an appropriate batched classification and a powerful cloud server, ProteiNN could provide better performance and be scalable with respect to the number of queriers.

4.7 Conclusion of privacy-preserving neural network classification

In this chapter, we have introduced three privacy-preserving neural network (NN) prediction solutions, namely PAC, SwaNN, and ProteiNN, that enable a cloud server to efficiently classify an input coming from a querier on a neural network model and obtain a sufficiently accurate prediction result without sacrificing the privacy of the underlying data.

In these solutions, we implement the privacy-by-design approach and consider two cases: (i) Whether the NN model is in the plaintext or encrypted form; and (ii) minimising the cryptographic technique(s) incompatibilities without disclosing data privacy and accuracy of the NN classification.

- PAC designs a neural network from scratch to be compatible with the cryptographic technique, namely 2PC, and therefore, PAC balances the trade-off between privacy, accuracy, and efficiency. As previously discussed, although 2PC supports nonlinear operations, and they can, therefore, increase the accuracy, 2PC suffers from the communication overhead. Moreover, in 2PC solutions, the client performs the same NN operations as the cloud server (i.e., a powerful model provider). In other words, the workload of the client is the same as the cloud server, but the computation resource may not be the same as the latter.

- SwaNN employs the best worlds, namely the additively homomorphic scheme, Paillier instead of fully homomorphic scheme thanks to the plaintext NN model in the cloud, and arithmetic and Boolean sharings instead of Yao's Garbled circuits. We have presented how one can easily support each underlying NN operation using these two schemes only. We have proposed a protocol to securely compute the Square function and employ several optimisations consisting of some data packing dedicated to the Paillier cryptosystem and the use of the multi-exponentiation algorithm to reduce the cost of multiplications to minimise the computational cost.

- The two previous solutions mainly consider a *one-to-one* scenario where one querier is querying the data, and one cloud server or a model provider is performing the NN classification operations. ProteiNN enhances this scenario to *one-to-many* scenario whereby a model provider outsources a NN model to a cloud server, encrypts its model, and wants to keep it encrypted on the cloud. The client wants to classify some input by using the encrypted model on the cloud and hiding the result from any party and thus encrypts its input with its public key. The cloud helps the model provider store the encrypted NN and apply it to some inputs coming from

the different clients. For this aim, ProteiNN employs a homomorphic proxy re-encryption and simple encryption to provide data privacy, minimise the model provider's workload, and maintain the accuracy of the NN prediction.

Chapter 5

Privacy-preserving Clustering

The best preparation for tomorrow is doing your best today.

H. Jackson Brown, Jr

In this chapter, we focus on the privacy concerns arising from the trajectory data analysis when employing cryptographic techniques. Further, we introduce a new privacy-preserving trajectory clustering solution based on 2PC.

5.1 Introduction

As introduced in Section 2.2, clustering is an unsupervised machine learning technique that partitions elements of a given dataset and groups them into some subsets, namely clusters, with respect to their similarities. With the approach of Machine Learning as a Service (MLaaS), the clustering tasks and hence the corresponding dataset are outsourced to a cloud server to address the need of the companies possessing a large number of data but not having sufficient computational resources and/or expertise in clustering techniques such as k-means or TRACLUS (see Section 2.2). However, because the data outsourced to the cloud server is usually privacy-sensitive, they need to be protected before their outsourcing. Therefore, companies could become compliant with the data protection regulation(s) while trying to infer valuable information out of their dataset. In order to preserve the privacy of the data and consequently their corresponding owners, we design a privacy-preserving protocol for TRACLUS.

Consider the scenario depicted in Figure 5.1 whereby a data owner (or a company) having collected multiple trajectories coming from multiple individuals would like to delegate the execution of the actual trajectory clustering algorithm called TRACLUS to

an untrusted but powerful cloud server. Since data is privacy-sensitive, the data owner must protect all the trajectory information before outsourcing them to the cloud server. The underlying information should be kept secret, but still be useful to obtain meaningful information from the protected them: Cryptographic techniques such as secure two-party computation (2PC, see Section 3.2), or homomorphic encryption (HE, see Section 3.3) can be employed to preserve the data privacy against any parties except its owner. Many researchers have studied data privacy while executing some clustering algorithm, and several privacy-preserving clustering solutions have been proposed in the state-of-the-art.

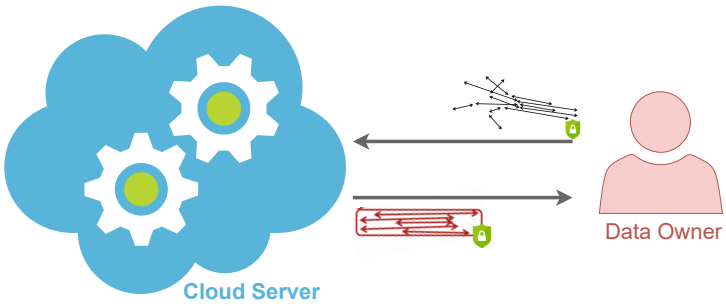


Figure 5.1 – Clustering in the single-server scenario

Yet, the trajectory clustering algorithm TRACLUS was not studied. In the following section, we introduce our solution for the privacy-preserving trajectory clustering problem, namely pp-TRACLUS: In our privacy setting, a data owner wishes to cluster its input (generally, a dataset consisting of line segments) and outsource this dataset and the TRACLUS clustering phase operations to a cloud server. At the same time, the data owner does not want to reveal its dataset and the corresponding result to the cloud server. Further, the cloud server would compute the clusters over the underlying dataset without discovering and disclosing any information.

5.2 Privacy vs. Clustering

This section identifies the challenges while combining the cryptographic technique(s) with the clustering algorithm(s).

As discussed earlier in Section 2.2, TRACLUS consists of two phases: the Partitioning phase and the Clustering phase. These phases mostly contain complex operations such as division, logarithm, square root or sine function. Furthermore, the TRACLUS phases need to be iterated as many times as possible to end up with some clustering of good quality evaluation. On the other hand, the dataset should be protected by some cryptographic technique. When considering the complex TRACLUS phases, their operations, and their integration with some cryptographic technique, we propose to design a solution for the clustering phase under privacy protection.

Our goal is to allow an untrusted third party named a cloud server to help a data owner (or a company being abundant of the massive amount of data coming from its clients/users) perform trajectory clustering and not to discover anything regarding the

data of the data owner. Thus, we propose a solution whereby a data owner possesses a set of data and wishes to make them into groups based on their similarities. However, this data owner does not have domain-specific expertise (and so the data owner cannot easily be confident about which clustering) and/or computational resources (powerful machines or a person who can run the underlying clustering technique) to group them. Therefore, some cloud server helps the data owner compute clustering without learning anything about the input data. Yet, this could raise serious privacy problems since the data being collected, stored, and processed is usually privacy-sensitive. Therefore, it is a must to implement cryptographic techniques such as homomorphic encryption or secure multi (or two)-party computation to be obedient with the data protection regulations such as the General Data Protection Regulations (GDPR) [2] or ePrivacy [3]. Although combining the cryptographic technique(s) with the clustering algorithm ensures data privacy, they result in some bottlenecks regarding the cost of computation and communication. They also might not easily and efficiently support complex operations in the clustering algorithm. Thus, when creating a privacy protection solution for clustering, the underlying privacy-preserving clustering solution should be built from scratch by taking the privacy-by-design approach into account.

We identify the following three challenges when guaranteeing data privacy with the assembling of cryptographic techniques and clustering techniques:

- **Challenge 1: Complex clustering operations.** k-means, DBSCAN, and TRACLUS consist of several complex operations such as division, square roots, sine, etc., and these underlying operations thus should be revisited and built employing some optimisation(s).
- **Challenge 2: Optimisation of parameters.** There are some clustering algorithm-specific parameters such as the number of clusters and the number of iterations to decide how many groups in the datasets when utilising k-means, or the parameters of $MinLns$, ϵ, and the number of iterations for TRACLUS. Such parameters have a non-negligible impact on the complexity of the clustering techniques and thus the quality evaluation for their results. These parameters should be selected carefully when designing the privacy-preserving variant of (trajectory) clustering algorithms. Therefore, privacy-preserving variants of the underlying clustering techniques should not sacrifice too much clustering quality evaluation.
- **Challenge 3: Real numbers instead of integers.** Although the clustering algorithms are run mostly over real numbered dataset(s), integers are usually supported by cryptographic techniques. Enabling floating point number arithmetic or transforming them into integers thus is needed when using the underlying cryptographic technique. Yet, transforming these numbers into integers can have a non-negligible impact on the clustering quality evaluation.

The complex clustering algorithms' operations should be customised to reduce the overhead resulting from developing the privacy-preserving variants of the clustering algorithm. Such customisation should not have an impact on the actual quality evaluation and efficiency of the clustering algorithm and do not cause any loss in data privacy.

5.3 Prior Work

The problem of privacy-preserving clustering has been firstly investigated by data science researchers in a setting in which data is partitioned horizontally, vertically, and arbitrarily. Then, cryptography researchers started to propose solutions for this topic thanks to recent advances in cryptographic techniques.

5.3.1 *k*-means

For privacy-preserving k-means solutions, the work described in [142] proposes a solution based on vertically partitioned data among two parties. The horizontally partitioned data has been studied in [142–145]. Finally, the rest of the studies we reviewed [146, 147] focus on the scenario of arbitrarily partitioned data consisting of a combination of the vertical and horizontal partitioned dataset.

Solutions in [142, 143, 146, 147] make use of 2PC. In this approach, two parties cooperatively execute the k-means algorithm on their joint datasets. The second approach adopted by authors of [145] delegates the computational burden of k-means over a dataset owned by a single or multiple parties to an untrusted cloud server by utilising FHE: The cloud does not learn any information about the data and the clusters.

Most of the solutions studied in privacy-preserving k-means [142,143,146–148] employ the secure function evaluation techniques such as the Paillier encryption scheme, Yao's protocol, and special secure function for addition, multiplication, and division. Some of them invoke the Paillier homomorphic scheme [147,149]. Only one work proposes a TFHE-based solution [145]. We highlight that several proposed solutions such as [142, 146] leak some information about intermediate cluster values, and the requirement of non-colluding parties is ignored.

A very recent solution [150] propose to use 2PC for the outsourced privacy-preserving k-means tasks, and this solution is assumed to be extendable to the MPC setting.

5.3.2 DBSCAN

The studies in [151, 152] consider a scenario whereby two data owners who want to jointly execute the DBSCAN algorithm over their private data: [151] may need for the existence of a trusted third party. Moreover, the proposal [152] leaks the information leakage about the clusters and their cardinality, and [151], similar to [152] leaks the size of clusters. [149] propose a study based on additive masking and HE to preserve data privacy, and data is vertically partitioned. The protocol uses FHE, but any performance evaluation is not applied.

Moreover, two privacy-preserving DBSCAN protocols are proposed over horizontally and vertically partitioned data in [153] who can extend the solution to the arbitrarily partitioned data. Authors make use of Yao's garbled circuits, but they leak the information of cluster sizes or values of parameters. Two solutions [154, 155] employ a differentially private mechanism to preserve data privacy while running DBSCAN. The adding noise to data affects the clustering quality result negatively.

5.3.3 Trajectory Analysis

Very few previous studies investigate the problem of private trajectory analysis. Private-Hermes [156] and Hermes++ [157] use anonymisation techniques to generate crafted, realistic fake trajectories to allow users to securely query mobility (trajectory) datasets. Other works such as [158–160] explore the problem of privacy-preserving ride sharing, which consists of finding a match between parts of trajectories. These solutions mostly use additively HE combined with private set-intersection.

A recent work [161] reviews the state-of-the-art solutions, re-implements four recent solutions, and compares them concerning the trade-off between privacy, efficiency, and quality evaluation results.

Note that no previous work has investigated private trajectory clustering. In the next section, we propose our privacy-preserving TRACLUS protocol, which is the first solution enabling private and efficient trajectory clustering technique, namely TRACLUS, between a data owner and a cloud server.

5.4 pp-TRACLUS: Privacy-preserving TRAjectory CLUStering

This section presents pp-TRACLUS, a privacy-preserving trajectory clustering solution [162]. This work was published in *ACM ASIACCS 2021, 16th ACM ASIA Conference on Computer and Communications Security*.

TRAjectory CLUStering (TRACLUS) based on DBSCAN is employed to group trajectories containing sequences of (multidimensional) line segments (see Section 2.2.3). Each trajectory can contain different numbers of line segments, and this can be a problem for some clustering algorithms such as k-means since k-means expects to take some numbered data items for the input. The application of TRACLUS can vary from the traffic analysis, location-based social networks to the vacation recommendation [163]; therefore, it can be attracted by some tourism agencies to obtain some results regarding when and how many people travelling to where without sacrificing the privacy of individuals and with taking advantage of Machine Learning as a Service and so the cloud computing. Therefore, the privacy-preserving variant of TRACLUS can play a crucial role in the analysis of location data collected by some data owner, and even this analysis usually outsourced and being compliant with the data protection regulations [2, 3] can be used for various applications such as controlling the spread of Covid-19 while ensuring data privacy. In order to detect the similar routes which are used by (infected) people during their visits and, at the same time, to preserve data privacy, we propose a solution for TRACLUS by utilising secure two-party computation while maintaining good clustering quality and efficient computation.

The next section introduces our design for the privacy-preserving TRACLUS based on 2PC.

5.4.1 pp-TRACLUS: Description

This section presents our privacy-preserving protocol for TRACLUS, pp-TRACLUS, which is the first solution enabling input (i.e. dataset) and result/output (i.e. clusters, etc.) privacy and keeping the clustering quality as same as the plaintext version of it.

In pp-TRACLUS, a data owner holds a dataset of trajectories and assigns the trajectory clustering operations over the underlying dataset to an untrusted cloud server. We assume that the data owner has already run the partitioning phase and would like to outsource the clustering phase by computing arithmetic shares of the line segments to the cloud server. Hence, the cloud server would perform the TRACLUS clustering phase in a privacy-preserving manner.

As described in Section 2.2.3 and illustrated in Figure 2.3, the tripartite TRACLUS distance metric *dist* involves some complex operations. For instance, *dist* involves divisions and sine computations which are relatively expensive in 2PC or HE. In order to ensure data privacy and execute the clustering phase in an efficient manner, we approximate this distance metric. As the Euclidean Distance (ED) is the most common metric that is used for trajectory clustering [164], we propose to replace the original TRACLUS distance metric with a combination of EDs because the perpendicular, parallel, and angular distances in Equations 2.14, 2.15, and 2.16 are taken into account when considering EDs between each pair of points of the two line segments L_i and L_j (i, j are two positive integers). We replace ED with the computation of its square, namely the Squared Euclidean Distance (SED) in Equation 5.1, which is a common computation to be easily compatible with 2PC [165], and thus results in more efficient computation results. Whether two line segments L_i and L_j are checked to be neighbours, while one checks whether $\sum \mathsf{ED}(L_i, L_j) \leq \epsilon$, within 2PC, one employs the computation of $\sum \mathsf{SED}(L_i, L_j) \leq \epsilon^2$.

$$\mathsf{SED}(L_i, L_j) = \sum_{r=1}^{N} L_{ir}^2 + 2 \prod_{r=1}^{N} L_{ir} \cdot L_{jr} + \sum_{r=1}^{N} L_{jr}^2 \tag{5.1}$$

where L_i and L_j are two line segments, and $r = 1, \ldots, N$ denotes the components of line segments L_i and L_j when L_i and L_j can be multidimensional.

Given L_i and L_j defined with starting points $s_i = (x_{si}, y_{si})$ and $s_j = (x_{sj}, y_{sj})$ and ending points $e_i = (x_{ei}, y_{ei})$ and $e_j = (x_{ej}, y_{ej})$, the approximated distance measure d_{pptrac} (depicted in Figure 5.2) for pp-TRACLUS is defined as follows:

$$d_{pptrac}(L_i, L_j) = \mathsf{SED}(s_i, s_j) + \mathsf{SED}(s_i, e_j) + \mathsf{SED}(e_i, s_j) + \mathsf{SED}(e_i, e_j) \tag{5.2}$$

In pp-TRACLUS, the data owner initiates the privacy-preserving protocol by creating the shares of line segments, ϵ^2, and $MinLns$ using AS . Share, and further by sending one share of them to the cloud server. The data owner and the cloud server interactively perform each operation of pp-TRACLUS: They compute the distance matrix using the shares of line segments and ϵ^2, and they compute the clusters using the shares of this

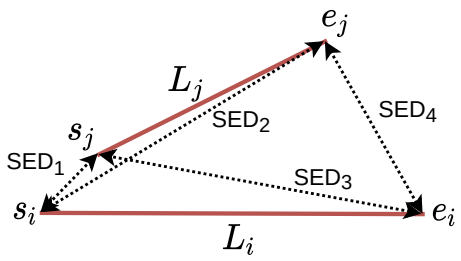


Figure 5.2 – Approximated distance measure d_{pptrac}

distance matrix and $MinLns$. Finally, each party obtains one share of the pp-TRACLUS clusters' result. The cloud server sends its share to the data owner who adds these shares to get the final result.

To summarise, during the distance calculations phase of pp-TRACLUS, the data owner and the cloud server jointly compute the four Squared Euclidean Distances between each pair of line segments and sum them to obtain the distance metric d_{pptrac}. This distance is further compared to the threshold ϵ^2 in order to check whether two line segments are neighbours.

5.4.2 Security Evaluation

In pp-TRACLUS, we assumed that the data owner and the cloud server are semi-honest where all parties in the pp-TRACLUS run are honest to correctly follow the pp-TRACLUS steps; however, they are curious and may try to gain as much information as possible from the data they receive. This means that they might store some data and later combine the underlying data to obtain more information. The goal of the data owner is to hide its input data, namely its dataset and its corresponding clustering result from the cloud server. We employ 2PC, particularly AS . Share or BS . Share, that achieves indistinguishability given that the shares are generated from a uniformly random distribution [132]. Thanks to the security of 2PC, pp-TRACLUS is secure and ensures the privacy of the underlying data while computing the privacy-preserving trajectory clustering.

5.4.3 Performance Evaluation

In this section, we evaluate the performance of our privacy-preserving clustering protocol based on TRACLUS and 2PC using three different datasets related to a scenario in where a data owner owns a dataset of line segments of several trajectories and wishes to learn similarities or some pattern among them with the help of some cloud server. Due to the nature of the dataset (i.e., usually privacy-sensitive), and thus before outsourcing, the data owner splits all line segments into two shares and sends one share of them to the cloud server. Then, the data owner and the cloud server run the pp-TRACLUS protocol over their shares. Once all operations are performed, the cloud server sends its share for the result to the data owner, and the data owner obtains the result by bringing the shares of the clusters' result.

Datasets

We employ several datasets to evaluate the performance of pp-TRACLUS.

- **Travel:** This "synthetic" dataset (not related to real individuals) is a 2-dimensional trajectory data corresponding to location of people's mobile phones. It has been created and provided by Orange S.A.[1], a major telecom provider, based on anonymised indicators of real trajectories. It consists of 40000 line segments.
- **Hurricane:** This dataset [7] contains 2-dimensional track data of Atlantic hurricanes from 1950 to 2006. It has 608 trajectories with 18343 line segments.
- **Deer:** This dataset [7] corresponds to 2-dimensional movements of deers in 1995. There exist 32 trajectories which correspond to 20033 line segments.

We highlight that Deer, Hurricane, and Travel datasets are already in integers; therefore, they do not need to be approximated.

Experimental Setup

To implement our solution, we use the ABY framework [42] written in C++, and the security level is set to 128-bit. The experiments are performed on two separate systems: one desktop equipped with 4.0 GHz Intel Core i7-7800X processor with Ubuntu 18.04.3 LTS and 128 GB RAM, and one cloud server with the properties of 2.30 GHz Intel Xeon Gold 6140 processor with Ubuntu 20.04 LTS and 64 GB RAM.

The pp-TRACLUS performance results are illustrated in Table 5.1. These results correspond to the average from the execution of 10 individual simulations. We observe that while the 2PC-based pp-TRACLUS, with the use of arithmetic sharing for the distance computation and Boolean sharing for the clustering computations, clusters the Travel dataset containing 400 line segments in 33.10 mins, it takes 34.28 mins for 400 line segments when using only Boolean sharing for both computations.

Table 5.1 – Performance evaluation for pp-TRACLUS on the Travel dataset with $\epsilon^2 = 4.5 \times 10^9$ and $MinLns = 3$

pp-TRACLUS	# of Line Segments	Time (mins)	Bandwidth (GB)
Arithmetic & Boolean Sharing	100	2.66	0.49/0.49
	400	33.10	10.27/10.27
	1000	173.10	93.53/93.53
Boolean Sharing	100	2.88	4.6/4.6
	400	34.28	75.6/75.6
	1000	210.58	501.16/501.16

Moreover, we compare the clustering quality evaluation based on the simplified TRACLUS distance measure d_{pptrac} and the original TRACLUS tripartite distance

[1] https://www.orange.fr

metric *dist* on our tested datasets. We conduct experiments with various values for ϵ^2 and $MinLns$ for the Hurricane dataset, the Deer dataset, and the Travel dataset (we name them Experiment 1 and Experiment 2 for each dataset). As in [7], the optimal values for ϵ^2 and $MinLns$ are computed with simulated annealing. The ϵ^2 values differ for TRACLUS and pp-TRACLUS because of the different distance measures. Note that all ϵ^2 for Experiments 1 and 2 result in the same entropy level in simulated annealing. As no ground truth is known for the trajectory datasets Hurricane, Deer, and Travel, we rely for this on well-established clustering quality indices [28–30]: SC, SC_{noise}, and DBCV as described in Section 2.2.4.

Table 5.2 depicts the results for the Hurricane dataset. In Experiment 1, TRACLUS outputs one cluster less than pp-TRACLUS. Moreover, the number of elements marked as noise and the number of clusters are larger with pp-TRACLUS than with the original plaintext TRACLUS. Nevertheless, we observe that pp-TRACLUS outputs better results with respect to SC, SC_{noise}, and DBCV than the original TRACLUS.

Table 5.2 – Clustering quality assessment for TRACLUS and pp-TRACLUS on the Hurricane dataset. For all scores larger values are better (best marked in bold).

Dataset	**Score**	**Experiment 1**		**Experiment 2**	
		TRACLUS	**pp-TRACLUS**	**TRACLUS**	**pp-TRACLUS**
Hurricane	**(ϵ^2, $MinLns$)**	$(24, 5)$	$(5000, 5)$	$(4, 5)$	$(2250, 5)$
	# of Clusters	2	3	11	13
	Noise	129	150	597	645
	SC	0.27	**0.87**	0.44	0.79
	SC_{noise}	0.27	**0.86**	0.42	0.76
	DBCV	0.72	**0.97**	0.64	0.76

Results for the Deer dataset are given in Table 5.3. The number of clusters created from the Deer dataset are equal with TRACLUS and pp-TRACLUS. When only one cluster is found, SC and DBCV cannot be computed. Nevertheless, having only one cluster does not necessarily mean that the quality of the clustering algorithm is low. It simply says that the algorithm found only one group of similar elements.

Table 5.3 – Clustering quality assessment for TRACLUS and pp-TRACLUS on the Deer dataset. For all scores larger values are better (best marked in bold).

Dataset	**Score**	**Experiment 1**		**Experiment 2**	
		TRACLUS	**pp-TRACLUS**	**TRACLUS**	**pp-TRACLUS**
Deer	**(ϵ^2, $MinLns$)**	$(400, 3)$	$(1 \times 10^6, 3)$	$(282, 3)$	$(550 \times 10^3, 3)$
	# of Clusters	1	1	2	2
	Noise	1	480	20	1333
	SC	N/A	N/A	0.089	**0.36**
	SC_{noise}	N/A	N/A	0.089	**0.34**
	DBCV	N/A	N/A	0.47	**0.79**

Tables 7.3 and 7.4 show the results of Experiments 1 and 2 of pp-TRACLUS and TRACLUS for the Travel dataset. We notice the same behaviour with respect to the quality evaluation metrics as for the Hurricane and Deer datasets shown in Tables 5.2

and 5.3. In Experiment 1, the number of input records marked as noise by pp-TRACLUS is larger than the number of elements marked as noise by the original plaintext TRACLUS. However, when fine-tuning the values of ϵ^2 and $MinLns$, it is possible to decrease the number of elements marked as noise (See Table 7.4 and Section 5.4 in [7]).

To summarise, our simplified and approximated distance measure d_{pptrac} tends to create a larger number of clusters (resulting in smaller clusters in average) and marks more elements as outliers. Nevertheless, our quality evaluation with well-established clustering quality evaluation gives even better results for the three analysed datasets showing that its quality evaluation is comparable to the original tripartite distance metric of TRACLUS.

5.4.4 Summary

We have presented the first privacy-preserving trajectory clustering protocol based on the TRACLUS algorithm and the use of 2PC named as pp-TRACLUS. TRACLUS is a DBSCAN-based scheme optimised for the trajectories clustering and contains many complex functions, for which we design and implement an approximated distance measure by proposing the use of the combination of Euclidean Distance and its square for the TRACLUS tripartite distance metric to efficiently integrate with 2PC. Moreover, with lessening the iteration number, namely the solution of pp-TRACLUS$'$, we could obtain more number of clusters, and this may be used as a solution for some applications such as the Covid-19 tracking, which needs more precision.

5.5 Conclusion of privacy-preserving clustering

This chapter has introduced a new privacy-preserving trajectory clustering solution, namely pp-TRACLUS containing a data owner and a cloud server, to efficiently group a dataset of line segments coming from a data owner without scarifying the underlying segments privacy.

pp-TRACLUS designs a privacy-preserving trajectory clustering solution from scratch by considering the privacy-by-design approach to be compatible with the cryptographic technique, namely 2PC: We propose to simplify the complex distance metric of TRACLUS to an approximated distance measure d_{pptrac} containing some Euclidean distances which take all possible location differences (i.e., the horizontal, vertical, and angular distances) among two line segments into account. Therefore, pp-TRACLUS balances the trade-off between privacy, clustering quality, and efficiency. In 2PC-based solutions, the data owner performs the same TRACLUS operations as the cloud server: The data owner's workload is the same as the cloud server, but the computation resource may not be the same as the cloud server.

Part II

Privacy-preserving two-server machine learning techniques

Chapter 6

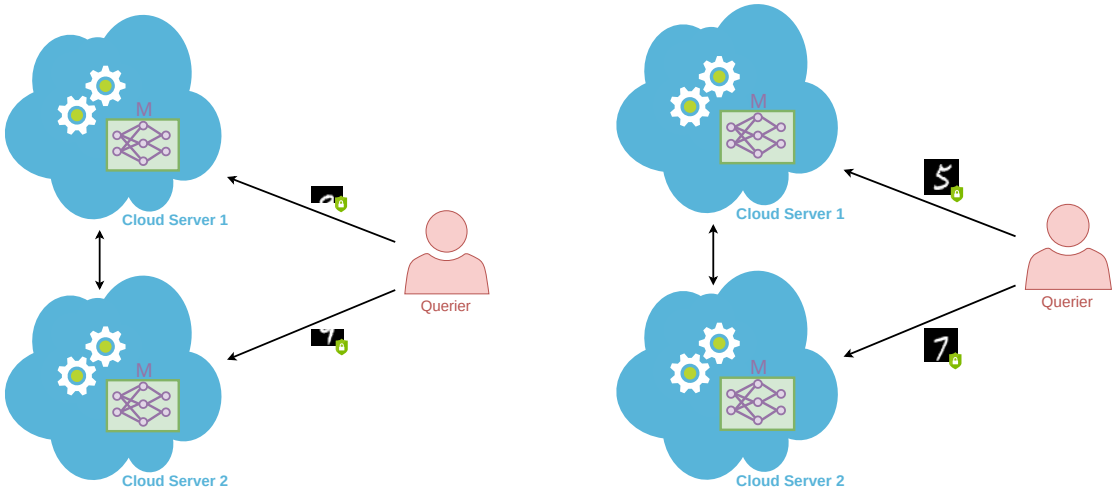
Privacy-preserving Neural Network Classification

In order to be irreplaceable one must always be different.

Coco Chanel

In the first part of this thesis, namely Chapters 4 and 5, we have described the design, development, and implementation of privacy-preserving neural network and trajectory clustering when operations are outsourced to a single cloud server. In the second part of this thesis, we investigate a scenario whereby two non-colluding cloud servers help querier(s) or data owner(s) execute privacy-preserving neural network classification, trajectory clustering, and data aggregation tasks. In the two-server scenario, we mainly aim to decrease the workload of the queriers/data owners when the underlying machine learning tasks are requested. Moreover, we also show that using different cryptographic techniques or the combination of them for designing neural network classification, trajectory clustering, and data aggregation tasks affects the data privacy, the performance evaluation (i.e., querier(s) or data owner(s) has/have little work to perform), and accuracy/quality evaluation of the proposed solutions in the first part of this thesis.

In this chapter, we revisit the designs for the privacy-preserving neural network (NN) classification while taking advantage of two cloud servers; and therefore, we decrease the computation load of queriers, and further, allow two non-colluding cloud servers to jointly perform the requested NN predictions.



(a) with 1 image (b) with 2 images

Figure 6.1 – Neural network classifications in the two-server scenario

6.1 Introduction

Consider a scenario where we employ two non-colluding and semi-honest cloud servers and name it the two-server scenario whereby a querier can either split its private input $\mathbf{X}$ into two shares and outsource one share to each cloud server as in Figure 6.1a or send one private input $\mathbf{X}$ to each cloud server as in Figure 6.1b. These two cloud servers hold the NN model $\mathbf{M}$, which can be the same or different, and perform all heavy computations, and the querier involves very little in the computation: The querier encrypts its input $\mathbf{X}$ (or two shares of its input $\mathbf{X}$) with its public key and sends the encrypted input $\mathbf{X}$ (or shared-encrypted input $\mathbf{X}$) to the two cloud servers, which jointly compute the private prediction result using the NN model $\mathbf{M}$. Similar to the solution described in Section 4.5, depending on the operation performed by one cloud server, the other one (instead of querier in Section 4.5) might be involved in the computations.

6.2 Prior Work

In this section, we briefly define the problem that we analyse and solve in two-server aided our solution and review the prior work.

Similar to Section 4.2, one encounters several challenges when developing a privacy-preserving NN prediction solution: (i) Size of NN which is defined as the number of layers, the number of neurons in these layers, and the sizes of input and output layers. Having a large size of them increase the complexity of NN, and this should be customised when combining NN with cryptographic techniques; (ii) Complex NN operations which cannot be easily supported by the employed cryptographic techniques, and therefore, they are needed to be optimised; and (iii) Real numbers which can be another challenge

to be addressed since mostly cryptographic techniques support integers.

The design for a privacy-preserving NN classification solution drew the attention of researchers, and several solutions that provide privacy protection in NN predictions are proposed (See Section 4.3).

We analyse privacy-preserving neural network solutions consisting of more than one cloud server which are based on the method of secure multi(two)-party computation (MPC/2PC). In [135], SecureML designs a privacy-preserving neural network training and classification method using 2PC, where queriers/clients secretly share their own private data among two non-colluding cloud servers. SecureML builds the model with the stochastic gradient descent method. Authors compute ReLU using Yao's GCs and implement *polynomial approximations* of nonlinear functions such as the *Sigmoid* and *Softmax* functions. Additionally, a solution for switching between arithmetic and Yao's sharings is proposed. As an extension to SecureML, authors [166] propose ABY^3, which shares the private inputs between three non-colluding cloud servers. To securely share sensitive data among three cloud servers, the authors redefine arithmetic, Boolean, and Yao's sharings of the ABY framework [42]. Moreover, a very recent scheme named SecureNN [167] uses secure three-party computation for the training and classification phases with convolutional neural networks using the MNIST dataset. SecureNN shares the input and output among two parties employing 2-out-of-2 arithmetic shares, and the third party joins the protocols during the online computation. CrypTFlow [168] complies TensorFlow and runs MPC protocols for deep NN using Intel's Software Guard Extensions. ASTRA [169] and FLASH [170] contain four parties with a malicious party focusing on the NN training based on secret sharing. Another work, namely Trident [171] is a four-party framework based on Boolean, arithmetic and Yao's sharings and supports many machine learning techniques including (C)NNs. Recently, Banners [172] proposes binarised NN classifications utilising replicated secret sharing.

6.3 Two-server SwaNN

In this section, we propose a new version of SwaNN utilising two-server for privacy-preserving neural network prediction tasks, that uses an additively HE cryptosystem, namely the Paillier cryptosystem [9], and combines it with 2PC. This work was presented as a poster in two different venues: *ICT.OPEN 2019*[1] and *PUT 2019*[2] in which it was awarded as "the best poster". Moreover, SwaNN was published in *SECRYPT 2020, 17th International Conference on Security and Cryptography* [94].

6.3.1 SwaNN: Description

In order to reduce the workload of the querier, we design SwaNN in a two-server scenario where two semi-honest non-colluding cloud servers (CS1 and CS2) perform the computations together. The querier splits its input and private key, provides one share

[1] https://ict-research.nl/ict-open/

[2] https://petsymposium.org/2019/workshop.php

of the input and private key to CS1 and CS2. Thus, the computations on the querier side are delegated to CS1 and CS2.

With a desirable scenario, the workload on the querier side can be reduced in the presence of two non-colluding cloud servers: In the two-server scenario, the querier outsources the input to both CS1 and CS2, and CS1 and CS2 perform the operations and return the result to the querier. However, if only a single input is provided to the CS1 and CS2, one of them is going to be idle during the non-interactive phase of the computations. Thus, we propose to provide one different input to each cloud server to fully utilise the computation capabilities of the cloud servers and classify two inputs at once.

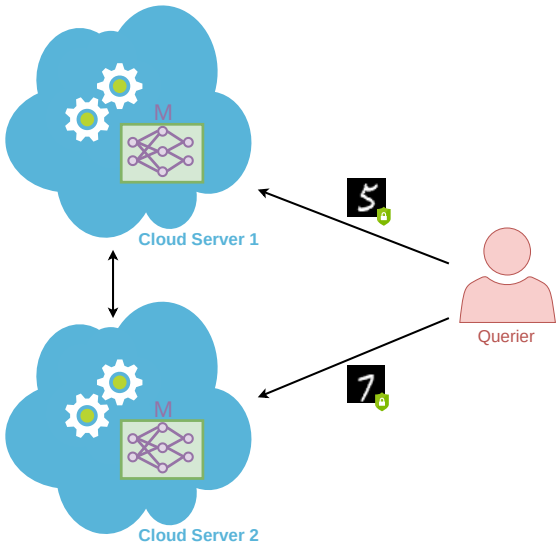


Figure 6.2 – Two-server scenario with two input inputs in SwaNN

Figure 6.2 illustrates our scenario. The querier encrypts two inputs with its public key and provides one input to each cloud server. Furthermore, the querier creates shares of the private key using $\mathsf{AS}.\mathsf{Share}$ (i.e., $\langle \mathsf{sk} \rangle_1 + \langle \mathsf{sk} \rangle_2 \leftarrow \mathsf{AS}.\mathsf{Share}(2, \mathsf{sk})$) for each cloud server as described in [54] and sends one share (i.e., $\langle \mathsf{sk} \rangle_i$, $i = 1, 2$) to each cloud server.

We present the common layers for any neural network and their functions in these layers in Section 2.1. Similar to the single-server scenario described in Section 4.5, we divide the computations into two phases as non-interactive and interactive phases:

Non-interactive phase

Similar to Section 4.5.2, when the two cloud servers receive the encrypted input compute the linear NN layers:

Convolutional Layer. The dot product, the operation in the convolutional (`Conv`) layer, is performed over a given input $\mathbf{X}$ encrypted with the Paillier encryption scheme [9] and a weight matrix $\mathbf{W}$ being plaintext on the cloud thanks to the homomorphic property

of the underlying encryption scheme. Note that each cloud server can perform this layer without the help of the other one since the decryption is not needed for the `Conv` layer.

Fully Connected Layer. The Fully Connected (`FC`) layer is composed of the matrix multiplication of a weight matrix $\mathbf{W}$ being plaintext and the encrypted result of the previous layer $\mathbf{X}'$. Remind that the operation in the matrix multiplication is dot product needed to be performed over each column and row pair, differently from the `Conv` layer.

Scaled Average Pooling Layer. This layer requires the sum of the values in the submatrix of the encrypted matrix result of the previous layer with omitting the division by the submatrix size. The additive homomorphism property of the Paillier cryptosystem computes this layer without any interaction.

Interactive phase

While the non-interactive phase can be performed by each cloud server locally, the interactive phase requires the involvement of both cloud servers. The two cloud servers can execute this phase sequentially based on a predetermined order or execute it in parallel, which improves the computation cost even further.

Activation Layer. The activation layer is also similar to the description in Section 4.5.2, yet it differs in the decryption procedure. In the single-server scenario in Section 4.5, the querier is responsible for performing the decryption operations. Nevertheless, in the two-server scenario, the decryption procedure is delegated to Cloud Server 1 ($\mathsf{CS1}$) and Cloud Server 2 ($\mathsf{CS2}$) along with their shares of the secret key sk. Therefore, the Decrypt algorithm in Protocol 1 is performed by both cloud servers. Protocol 4 illustrates how secure square protocol works when the computations are delegated to $\mathsf{CS1}$ and $\mathsf{CS2}$ which are assumed to not collude.

Protocol 4 Secure Square Protocol in the two-server scenario

Cloud Server 1 $(\mathsf{pk}, \mathsf{sk}_1)$		**Cloud Server 2** $(\mathsf{pk}, \mathsf{sk}_2)$
		$[x], r \in_R \{0,1\}^{\ell+\kappa}$
		$[x_r] \leftarrow [x] \cdot [r]$ (this is equal to $[x+r]$)
		$[x_r]' \leftarrow \text{Decrypt}([x_r], \langle \mathsf{sk} \rangle_2)$
	$\xleftarrow{[x_r]'}$	
$x_r \leftarrow \text{Decrypt}([x_r]', \langle \mathsf{sk} \rangle_1)$		
$x_r^2 \leftarrow x_r \cdot x_r$		
$[x_r^2] \leftarrow \text{Encrypt}(x_r^2, \mathsf{pk})$		
	$\xrightarrow{[x_r^2]}$	
		$[x_r^2] \cdot \left([r^2] \cdot [x]^{2r}\right)^{-1}$
		$[x^2] \leftarrow \left[x_r^2 - r^2 - 2xr\right]$

In Protocol 4, to compute the square of the encrypted input $[x]$, $\mathsf{CS2}$ randomly chooses a number $[r]$ from $\mathbb{Z}_N$, adds it to $[x]$ to perform a secure decryption using its share of secret key $\langle \mathsf{sk} \rangle_2$, and then sends it to $\mathsf{CS1}$. $\mathsf{CS1}$ decrypts $[x_r]$ once more and takes the

square of it. After square computation, CS1 encrypts using the public key pk of the querier and sends it back to CS2. Then, CS2 subtracts the underlying random values in the protocol to get the square of $[x]$.

We also use ReLU as an activation function, and CS1 and CS2 interactively perform ReLU thanks to the comparison gate of Boolean sharing, BS . Share.

Max Pooling Layer. Similar to the Max pooling layer in Section 4.5.2, we do not employ any approximation for Max. Further, the switching phases for computing Max are computed similar to the description in Section 4.5.2, but the decryption procedure is different like in Secure Square Protocol 4. Therefore, the Decrypt algorithm in Protocol 2 is performed by non-colluding CS1 and CS2.

6.3.2 Security Evaluation

SwaNN aims to compute neural network predictions under the privacy preservation assumption in the semi-honest adversarial model where the parties do not collude, and parties in the computation exactly follow the protocol steps. However, the querier and the two non-colluding cloud servers are curious to obtain some information from outputs of the computations and intermediary messages. We assume that the semi-honest adversary is non-adaptive and computationally bounded. The querier's goal is to hide the input image and its corresponding classification result from the two cloud servers. On the other hand, the two cloud servers do not want to reveal the NN model parameters used during computations to the querier.

We achieve our security goal thanks to the security of the cryptographic techniques we use in the design of SwaNN: In the non-interactive phase, the data privacy is ensured by the semantical security of the Paillier cryptosystem under the decisional composite residuosity assumption [9]. Therefore, the two non-colluding cloud servers CS1 and CS2 cannot discover any information about the underlying input of the queriers. Unlike the non-interactive phase, the NN layers such as x^2 require interactive computations between CS1 and CS2. For this computations, we employ 2PC, particularly AS . Share and BS . Share, that achieves indistinguishability given that the shares are generated from a uniformly random distribution [132]. Note that the Paillier cryptosystem and 2PC are secure, the security of the interactive phase of SwaNN can be deduced to the security of switching or decryption operations. Similar to the security of SwaNN in the single-server scenario 4.5.3, SwaNN in the two-server scenario is secure and does not leak any private information about the inputs, the corresponding results, and the NN model parameters.

Theorem 6.3.1. *The switching protocol π, i.e. Protocol 2, performed by CS1 and CS2, securely computes the functionality $f(\perp, [x]) = (\langle x \rangle_1, \langle x \rangle_2)$ in the presence of semi-honest, non-adaptive, computationally bounded adversaries.*

Proof. Theorem 6.3.1 for a corrupted CS1 and CS2 separately can be proved in a similar way of the proof of Theorem 4.5.1 instead of the client and the cloud server. Thus, we refer Section 4.5.3 for the details. □

6.3.3 Performance Evaluation

This section presents the performance evaluation of SwaNN in the two-server scenario whereby a querier is responsible only for encryption and the final decryption procedures and the two cloud servers compute all NN operations, and the comparison of our results with the state-of-the-art solutions. We used the C++ programming language for the implementation and the GMP 6.1.2 library for big integer operations. We used the ABY framework [42] for the 2PC operations. For the homomorphic operations, we used the Paillier implementation of ABY due to its efficiency. We selected 2048 bits modulus size for the Paillier operations to meet the current security standards. For the ABY operations, we selected 32-bit shares. The machine we used in the experiments runs Ubuntu 16.04 operating system with Intel Core i5-3470 CPU 3.20 GHz.

SwaNN in the two-server scenario adapts several optimisations methods for the Paillier encryption scheme and 2PC to minimise the computational cost and communication overhead during privacy-preserving NN predictions similar to Section 4.5.4: Data packing for the `Act` Layer, the Lim-Lee multi-exponentiation algorithm for the `Conv` and `FC` layers, and SIMD for 2PC operations.

Experiments

We implement SwaNN in the two-server scenario in three experiments using x^2 and ReLU as activation functions.

Neural Network Structures. The underlying experiments use the following NN structures: (i) one from CryptoNets [107], and MiniONN [97] is used to classify images of the MNIST dataset. This network in our experiment contains 2 `Conv`, 2 activation of x^2, 2 Scaled Mean pooling, and 2 `FC` layers, and achieves 98.95% of accuracy; (ii) one from MiniONN [97]. The structure is similar to the first network, yet the activation function is ReLU, and Max pooling is used in the `Pool` layer. Its accuracy of the network is 99.31%; and (iii) the last (and deeper) NN structure trained by [97] (in Figure 13 in MiniONN) performs image classifications on the Cifar-10 dataset. Its structure consists of 7 convolutional, 7 ReLU, 2 Max pooling, and 1 FC layer. The accuracy of this network 81.61%.

Experiment 1. Table 6.1 demonstrates the performance of SwaNN in the only-PHE and hybrid settings for each layer of CNN described in CryptoNets [107]. For the only-PHE setting, we provide the timings with and without optimisations. For the hybrid setting which employs PHE and 2PC, we provide only optimised timing values.

SwaNN in the two-server scenario with a slight increase in computation time can simultaneously process two input, particularly images. More particularly, in an optimised hybrid setting, the two cloud servers can compute the prediction result for two images in 10 seconds simultaneously.

In Table 6.2, we provide the benchmarking for the Square activation function by switching between PHE and 2PC for the computation of packing, decryption and unpacking operations. In the two-server scenario, both cloud servers spend approximately 6 seconds for the computation of the activation layer of two images.

Table 6.1 – Computation time per layer in the two-server scenario (in ms). The timings are provided for optimised and non-optimised PHE-only setting and optimised hybrid setting. The total timings marked with * show the simultaneous run time of SwaNN for two images.

	Non-optimised - PHE only		Optimised - PHE only		Optimised - Hybrid	
Layer	CS1	CS2	CS1	CS2	CS1	CS2
`Conv`	1883	1883	917	911	919	900
`Act`	33442	33319	23973	23941	2947	2984
`Pool`	34	34	34	33	33	34
`Conv`	2911	2948	1347	1344	1378	1364
`Pool`	37	37	38	37	37	39
`FC`	6579	6536	3802	3818	3973	3977
`Act`	3993	4009	2795	2797	607	573
`FC`	10	10	11	10	11	11
Total (ms)	48892*		32902*		9904*	

Table 6.2 – Detailed computation time for the activation layer in the two-server scenario for the hybrid setting (in ms).

Operation	CS1	CS2
Packing	413	406
Decryption	147	146
Unpacking	0.1	0.1
ABY	28	28
Encryption	2373	2685
Total (ms)	3265*	

Apart from the computation time, we also analyse the bandwidth usage of SwaNN for different settings. Table 6.3 presents the communication cost in both scenarios for the only-PHE setting and the hybrid setting. The packing technique used in the activation layers helps reduce the bandwidth usage by half. Besides, due to the interactive nature of 2PC, the bandwidth usage in the hybrid setting is higher than the only-PHE setting for the single-server scenario and the two-server scenario.

Table 6.3 – Bandwidth usage of SwaNN in different settings (in MB).

	Single-server	Two-server
PHE only (w/o opt.)	0.97	0.96
PHE only (w/ opt.)	0.51	0.51
Hybrid (w/ opt.)	1.69	1.69

Table 6.4 compares SwaNN and two state-of-the-art solutions: one FHE-based solution, CryptoNets [107] and one 2PC-based solution, MiniONN [97]. According to [107], CryptoNets requires 297.5 seconds for one prediction. MiniONN computes one prediction

in 1.28 seconds [97] (Note that the querier in MiniONN performs the same operations as the cloud server, and thus it can need the same facilities (e.g. expertise in machine learning and computation resources) as the cloud server already has). However, this computation requires 47.6 MB bandwidth usage. SwaNN computes the two images prediction in 10 seconds with the help of two cloud servers. Although the computation time of SwaNN is higher than MiniONN, SwaNN achieves a 28-fold less bandwidth usage, and the querier performs very little work: Encryption of the input, splitting the secret key into two shares, and the decryption of the encrypted result.

Table 6.4 – Comparison with the state-of-the-art in Exp. 1.

	Computation time (s)	Bandwidth usage (MB)
CryptoNets [107]	297.5	372.2
MiniONN [97]	1.28	47.6
SwaNN	9.9	1.69

Experiment 2. In the second experiment, we benchmark the performance of SwaNN with the ReLU activation function for the CNN model for MNIST described in [97]. We use maximum operation for pooling layers. We provide the timings for the Max pooling along with the ReLU function since we implemented them together. We measure the timings in the single-server scenario and the two-server scenario only with optimisations. Table 6.5 details the computation time for each layer.

Table 6.5 – Computation time per layer in the single-server and the two-server scenarios (in ms).

Layer	Querier	Cloud Server	Cloud Server 1	Cloud Server 2
`Conv`	–	10192	10196	10195
`Act+Pool`	6852	2593	11968	10613
`Conv`	–	1148	1150	1153
`Act+Pool`	778	467	1448	1411
`FC`	–	1325	1332	1360
`Act`	274	508	801	866
`FC`	–	5	5	6
Total (ms)	24242		26099*	

The computation cost is 24 seconds in the single-server scenario whereas for the two-server scenario, it becomes 26 seconds for two images since each layer has the larger inputs' size. Note that in the two-server scenario, the querier only does perform the encryption of the input image and the final decryption of the corresponding result. Therefore, the querier does spend any time for each layer during the classification of its input.

Table 6.6 compares SwaNN and MiniONN. While MiniONN outperforms SwaNN in computation time, SwaNN is more efficient with bandwidth usage in terms of communication cost.

Table 6.6 – Comparison with the state-of-the-art in Exp. 2.

	Computation time (s)	Bandwidth usage (MB)
MiniONN [97]	9.32	657.5
SwaNN	26.00*	160.9

Experiment 3. In the third experiment, we measure the performance of SwaNN with the ReLU activation function for a deeper neural network model for CIFAR-10 [97].

Table 6.7 – Comparison with the state-of-the-art in Exp. 3.

	Computation time (s)	Bandwidth usage (MB)
MiniONN [97]	544	9272
Gazelle [123]	12.9	1236
SwaNN	394.1	1939

In Table 6.7, we compare the performance of SwaNN with MiniONN and Gazelle. Clearly, SwaNN outperforms MiniONN in computation time and bandwidth usage. Nevertheless, Gazelle seems better than SwaNN. It is worth to note that these results are taken for the reference paper [123] and were hard to reproduce in our environment. Furthermore, in SwaNN, compared to Gazelle, we make use of simple mechanisms such as Boolean sharing (BS . Share) for ReLU and Max pooling instead of Garbled Circuits.

6.3.4 Summary

We have designed SwaNN in the two-server scenario for the NN classification tasks based on PHE and 2PC. On the one hand, we succeed in reducing the computational and communication costs of queriers and delegate those costs to the two non-colluding cloud servers, and, on the other hand, to ensure the privacy for the input data, its result, and the NN model. For this respect, we also have investigated minimising the computations at the cloud servers side and the overall computational cost for the proposed NN models.

6.4 Conclusion of privacy-preserving neural network classification

In this chapter, we have shown that a privacy-preserving neural network prediction solution in the two-server scenario is as secure and accurate as privacy-preserving neural network prediction solutions in the single-server scenario. Nevertheless, we have gained a significant impact on reducing the queries' workload during the classification, and they do not need to perform any computations rather than their input encryption and the corresponding results' decryption. We have employed two non-colluding cloud servers to compute all heavy operations of the neural network while private inputs are predicted without disclosing any information.

Chapter 7

Privacy-preserving Clustering

Create the highest, grandest vision possible for your life because you become what you believe.

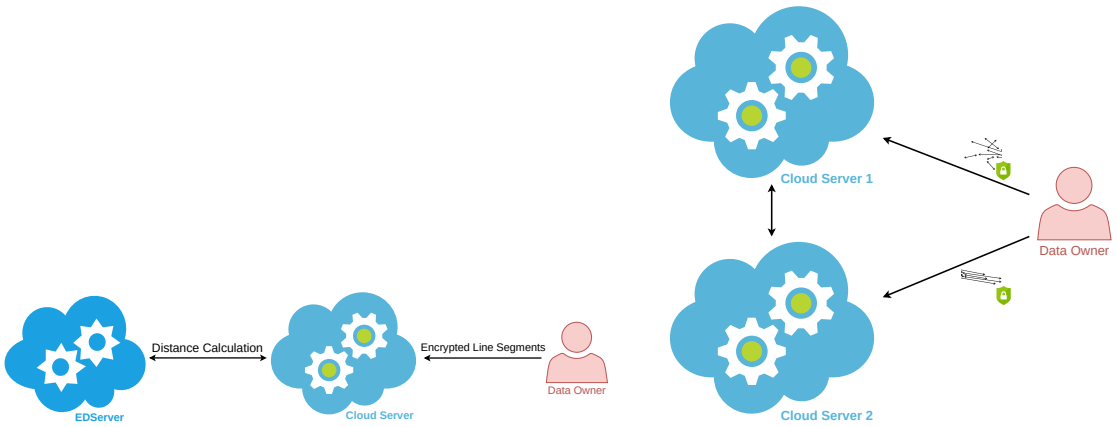
Oprah Winfrey

This chapter investigates the privacy-preserving clustering solutions utilising two non-colluding cloud servers to decrease the computation burden of the data owner(s). Further, we present two solutions employing two different cryptographic techniques and show the impact of the different cryptographic techniques on privacy, efficiency, and quality evaluation.

In order to ultimately benefit from the advantage of cloud servers and lower the workload of the data owners who outsource their dataset (and the clustering operations) to two non-colluding cloud servers and investigate the suitability of cryptographic techniques, we revise designing a privacy-preserving protocol for the trajectory clustering technique in Section 5, namely TRACLUS, with the help of these two cloud servers: We propose two solutions for TRACLUS under privacy preservation: One is based on the use of the Paillier cryptosystem, and the other one is based on 2PC. Therefore, we decrease the workload of data owners and allow these two cloud servers to perform TRACLUS operations without any interaction from data owners.

7.1 Introduction

We consider two scenarios whereby a data owner can send one private dataset D to the cloud server as in Figure 7.1a and the other cloud server helps the fist cloud server, or split its private dataset D into two shares and outsource one share to each cloud server



(a) pp-TRACLUS based on PHE (b) pp-TRACLUS based on 2PC

Figure 7.1 – Clustering in the two-server scenario

as in Figure 7.1b. These two cloud servers perform all heavy computations of TRACLUS, and the data owner involves very little in the computation: Encryption or arithmetic sharing.

7.2 Privacy vs. Clustering

This section briefly presents the problem of the clustering technique, which is outsourced together with the dataset (or datasets) to two non-colluding cloud servers.

Similar to Section 5.2 that presents the challenges in the single-server scenario, the two-server scenario has the same two challenges while ensuring data privacy with the integration of cryptographic techniques with clustering techniques: (i) Clustering algorithms contain complex operations, which are not easily and efficiently compatible with the cryptographic techniques; and (ii) the suitable parameters should be selected because they mainly affect the clustering quality evaluation.

This chapter describes a new scenario called a two-server scenario whereby the dataset(s) and the clustering algorithm are efficiently performed by two untrusted, non-colluding, and semi-honest cloud servers without any interaction between client and two-server. In this scenario, privacy-preserving clustering tasks should preserve data privacy, cluster privacy, and (full) privacy of the data processing, i.e., no intermediate data or operations should not be disclosed to any party, including the two semi-honest cloud servers.

Moreover, selecting the cryptographic technique can be crucial to obtain a private, efficient, and qualified clustering evaluation clustering technique. Cryptographic techniques, unfortunately, introduce some non-negligible overhead in terms of computational and communication costs. For example, a partially homomorphic encryption (PHE) scheme supports only additions or multiplications and incurs less expensive computation cost when compared to fully homomorphic encryption (FHE) schemes; yet, FHE is flexible to enable any linear operations to be processed over the encrypted data, but it is more

expensive than PHE, or while MPC/2PC is efficient in terms of computational costs and supports nonlinear operations over protected data, it is more expensive than an FHE scheme in terms of communication cost.

In order to reduce the overhead of the privacy-preserving clustering algorithm and meet the privacy requirements, one should propose the customised privacy-preserving designs, and the underlying designs should not have a non-negligible impact on the actual quality and performance evaluation of the requested clustering algorithm.

7.3 Prior Work

In this section, we investigate the privacy-preserving clustering solutions that contain more than two parties namely the data owner and the cloud server.

Privacy-preserving collaborative k-means is considered in [173–175]. In these studies, several parties cooperatively execute the k-means algorithm on their joint datasets. Vaidya et al. [173] propose the first privacy-preserving k-means solution based on the secure permutation of Du-Attallah [176] and Yao's Garbled Circuits, in which the vertically split data coming from three parties are aggregated to run k-means. However, this solution does not mention whether these three parties are non-colluding. The study proposed by Doganay et al. [174] can be considered as the extension of [173] with four parties to minimise the communication and computation costs and use the Paillier-based add-and-permute protocol. [177] clusters users in a social network utilising the Paillier and DGK encryption schemes [9, 178] and a secure comparison based on Yao's sharing.

Some solutions [179, 180] consist of multiple data owners who outsource their dataset to some (non-colluding) cloud server(s). [179] and [180] make use of a Chinese Remainder Theorem [181] based secret sharing over arbitrarily partitioned data and the Paillier cryptosystem over horizontally partitioned data, respectively. [182] propose a PHE-based privacy-preserving k-means protocol (accelerates the study of [183]) on the social networks. This protocol contains three parties: users (or data owners), a cloud server, and a user helper randomly selected among users and employs some optimisation techniques such as data packing for efficient computations. Note that users in [177, 182] join the secure k-means computations.

A recent solution [184] proposes an FHE-based privacy-preserving k-means protocol with the existence of two non-colluding cloud servers and uses data packing to provide efficient FHE computations. Due to the encoding on the plaintext dataset, this solution deviates the quality evaluation of the original k-means.

[185, 186] aim for a privacy-preserving DBSCAN protocol based on HE [9, 85] with the presence of multiple data owners and an untrusted cloud server, but leaks some information about the size of the final clusters.

While the privacy-preserving k-means algorithm is highly investigated, the proposal on the privacy variant of DBSCAN is very few, and any solution on TRACLUS has not been proposed yet in the state-of-the-art. Moreover, the presented solutions for clustering techniques rely on the use of either MPC or FHE, and unfortunately, the workload of the data owners in these solutions is heavy since usually multiple data owners (instead

of the data owner(s) and the cloud server(s)) jointly do perform the required clustering technique. Therefore, in the next sections, we propose two solutions that lower the workload of the data owners and delegate this workload to two non-colluding cloud servers.

7.4 Two-server pp-TRACLUS

This section presents our privacy-preserving trajectory clustering solutions in the two-server scenario. One based on the Paillier cryptosystem, was an initial research solution for pp-TRACLUS. The other one, which is based on 2PC [162], was published in *ACM ASIACCS 2021, 16th ACM ASIA Conference on Computer and Communications Security*.

In this section, we show that when pp-TRACLUS can be useful for real-time use case scenarios; for example, monitoring the people travelling from one place to another place and giving some results regarding their travels without putting danger their data privacy, and further, with keeping the clustering quality evaluation as good as the plaintext TRACLUS and with obtaining the good performance evaluation. Therefore, we propose two solutions for TRACLUS: We utilise two cloud servers to lower the workload of the data owner and to demonstrate the suitability of the selected cryptographic techniques.

7.4.1 Problem Statement

As introduced in Section 2.2.3, the goal of trajectory clustering is to find similar travelling routes in a set of trajectories. This requires clustering the trajectories based on the location information. Since this can be a computationally expensive operation and be needed for the expertise in machine learning, we consider a scenario where a data owner, having collected multiple trajectories (i.e., a dataset), wishes to delegate the execution of the actual TRAjectory CLUStering (TRACLUS) algorithm to two untrusted but powerful cloud servers. Therefore, all trajectory information is needed to be protected before their outsourcing to the two cloud servers. As discussed in Section 7.2, the dataset can be protected by utilising cryptographic techniques. However, the combination of TRACLUS and cryptographic techniques is not straightforward since TRACLUS involves several complex operations, and those require to be simplified in order to be supported by cryptographic techniques. Having a simplified version of TRACLUS may not be adequate since the cryptographic technique one chose to employ has a critical role in obtaining an efficient and high clustering quality evaluation. Therefore, in this section, we investigate the suitability of the cryptographic techniques when integrating them with TRACLUS, and also we show how we lessen the workload of the data owner(s).

The next two sections introduce our two-server aided designs for privacy-preserving TRACLUS based on the additively homomorphic Paillier encryption scheme and 2PC.

7.4.2 PHE-based pp-TRACLUS: Description

We propose a privacy-preserving trajectory clustering that combines the Paillier encryption scheme [9] and TRAjectory CLUStering (TRACLUS) [7] to find similar travelling routes.

We consider a scenario whereby a data owner, having collected multiple trajectories partitioned into line segments (Note that the partitioning phase with the original TRACLUS distance metric is assumed to be already performed by the data owner(s)), wishes to delegate the execution of the actual TRACLUS to an untrusted but powerful cloud server. Therefore, all trajectory information is needed to be protected before their outsourcing to the cloud server. TRACLUS mainly consists of a number of distance calculations (See Section 2.2.3). In order to perform these distance computations with the Paillier encryption scheme, some operations need to be either transformed or approximated. Indeed, the Paillier encryption scheme supports only additions over the encrypted data and some scalar multiplication, whereby the scalar is in the cleartext. We, therefore, propose to simplify the original distance metrics defined in Section 2.2.3 and replace them with the simple Squared Euclidean Distance (SED) as given in Equation 7.1. Remind that the positional difference and directional difference are already included in the combination of Euclidean Distances as in the perpendicular, parallel, and angular distances of TRACLUS in Equations 2.14, 2.15, and 2.16.

$$\mathsf{SED}(L_i, L_j) = \sum_{r=1}^{N} L_{ir}^2 + 2\prod_{r=1}^{N} L_{ir} \cdot L_{jr} + \sum_{r=1}^{N} L_{jr}^2 \tag{7.1}$$

where L_i and L_j are two line segments, and $r = 1, \ldots, N$ denotes the components of line segments when line segments L_i and L_j can be multidimensional.

Moreover, we propose an approximated distance measure d_{pptrac} for pp-TRACLUS (illustrated in Figure 7.2) where given two line segments L_i and L_j defined with starting points s_i and s_j and ending points e_i and e_j as in Equation 7.2:

$$d_{pptrac}(L_i, L_j) = \mathsf{SED}(s_i, s_j) + \mathsf{SED}(e_i, e_j) \tag{7.2}$$

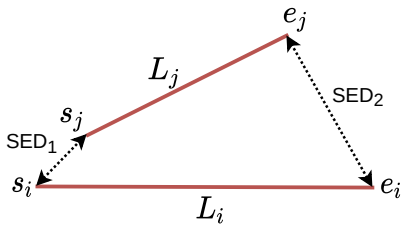


Figure 7.2 – Approximated distance measure d_{pptrac}

As shown in Equation 7.1, the computation of the Euclidean Distance involves some multiplication operations that the Paillier encryption scheme cannot inherently support. Therefore, we propose supporting multiplications with a newly designed protocol executed between the cloud server and another cloud server we name EDServer. The newly proposed solution is illustrated in Figure 7.3.

The data owner (DO) encrypts all line segments (e.g., the starting and ending points of all line segments when line segments are 2-dimensional) and their square with the Paillier

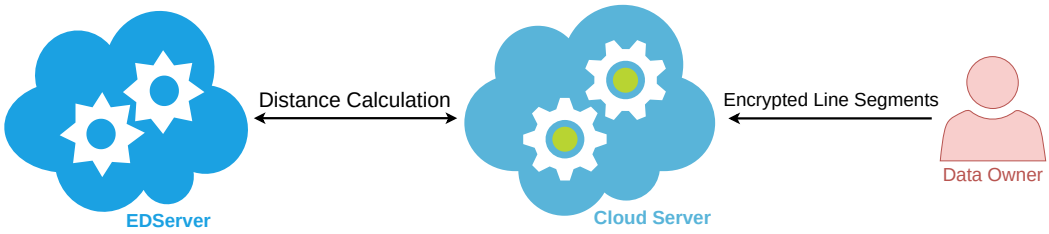


Figure 7.3 – pp-TRACLUS based on the Paillier cryptosystem

cryptosystem and sends them to the cloud server (CS). The encryption is performed using the public key of EDServer. Then, CS starts to compute SEDs between these segments: CS simply adds the encrypted squared coordinates depicted as the first and last terms in Equation 7.1. When multiplication between two encrypted numbers is needed, CS interacts with EDServer as follows: Let $[a]$ and $[b]$ be two encrypted numbers using EDServer's public key. In order to prevent EDServer from accessing the cleartext numbers, CS performs a scalar multiplication with some random numbers r_a and r_b to randomise the values and send these randomised results to EDServer. When these newly computed values are received, EDServer can decrypt them, perform the multiplication, and encrypt the multiplication result. CS can easily recover $[a \cdot b]$ thanks to its knowledge of the random numbers r_a and r_b as illustrated in Equation 7.3 (This calculation is similar to the Secure Square Protocol in SwaNN 4.5).

$$\begin{aligned}[a \cdot b] &= [a \cdot b] + [a] \cdot r_a - [a] \cdot r_a + [b] \cdot r_b - [b] \cdot r_b + r_a \cdot r_b - r_a \cdot r_b \\ &= [a + r_a] \cdot [b + r_b] - [a] \cdot r_a - [b] \cdot r_b - r_a \cdot r_b\end{aligned} \tag{7.3}$$

After having calculated all distances between one line segment and all other not clustered line segments, CS sends an array of all encrypted distances to EDServer, which can decrypt and determine which distances are smaller than a given threshold ϵ^2. As EDServer does not know which distance belongs to which pair of line segments, it only learns the number of not clustered line segments and how many elements belong to one cluster. However, as the size of each cluster may be published, this information might be considered as not a sensitive information leakage. EDServer denotes all positions in the array that contain a distance smaller than ϵ^2 with 1 and all others with 0. Once CS distinguishes which line segments are neighbours of the one that is analysed, it can create a cluster out of them. Then, it keeps to analyse each line segment in this cluster to determine whether they are also a core line segment of this cluster, and the other elements can be added. The process requires another SED calculation that can be run in the same way as previously described.

Security Evaluation

This solution is assumed to be based upon the semi-honest security model. In this security model, all parties, including the data owner, the cloud server, and EDServer, are honest to follow the protocol correctly. Still, they are curious to gain as much information as

possible while executing the privacy-preserving TRACLUS protocol. In other words, they might store information from old exchanged data and combine them to obtain more information. Moreover, we also assume that the cloud server and EDServer cannot collude. The goal of the data owner is to keep the dataset and the resulting clusters confidential to any parties including the cloud server and EDServer. Thanks to the semantically secure Paillier cryptosystem and randomisation, our privacy requirements are guaranteed.

Performance Evaluation

We present the experimental evaluation of PHE-based pp-TRACLUS using two public datasets, and further, we compare our solution with the PHE-based k-means solution.

Datasets. Two datasets are used for evaluating the performance of the proposed solution.

- **Hurricane**: This dataset [7] contains the track data of Atlantic hurricanes from 1950 to 2006. It has 608 trajectories with 18343 line segments.
- **Taxi Trip**: This dataset originally containing 800000 taxi routes was extracted from NYC Trip Sheet Data in January 2018[1]. We prepared this dataset consisting of 13511 trajectories for 265 users.

Note that the values of datasets for Hurricane and Taxi are already integers.

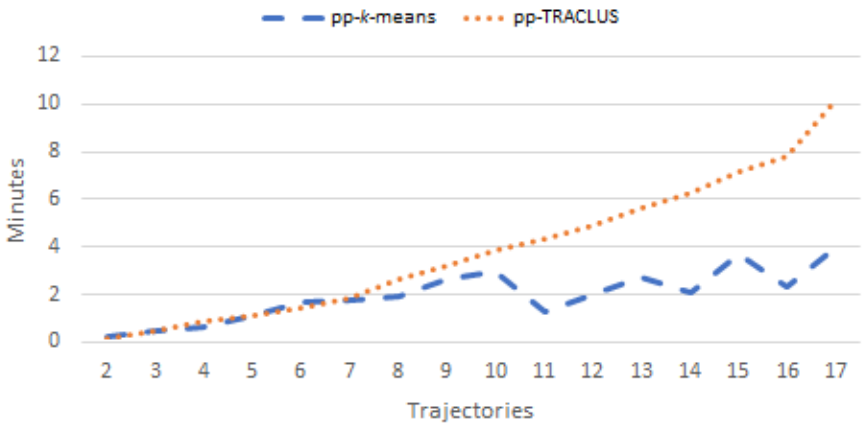


Figure 7.4 – Comparison of pp-k-means and pp-TRACLUS with Hurricane trajectories

Experimental setup. To implement the PHE-based pp-TRACLUS, we have used Python 3 library for the Paillier encryption scheme[2]. The protocol was carried out employing a laptop with 2.5 GHz Intel Core i5-2410M processor and 8 GB RAM.

[1]https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
[2]https://python-paillier.readthedocs.io/en/develop/

We have run several experiments for the privacy-preserving k-means (pp-k-means) of [182] and pp-TRACLUS over the Hurricane and Taxi datasets. For the Hurricane dataset, we use 17 trajectories of hurricanes, and the resulting timing can be found in Figure 7.4.

Moreover, we have timing results depicted in Figure 7.5 for 30 taxi trajectories created from the NYC Taxi dataset.

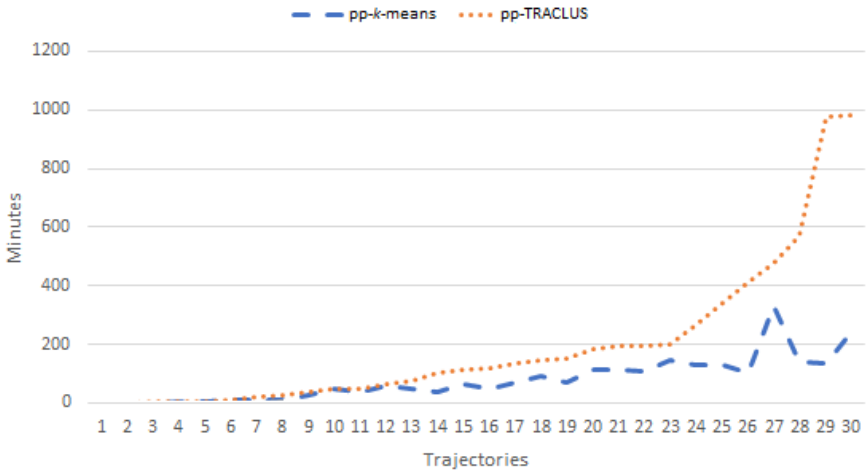


Figure 7.5 – Comparison of pp-k-means and pp-TRACLUS with Taxi trajectories

According to the performance evaluation, the privacy-preserving k-means solution is more efficient than pp-TRACLUS since one of the reasons is that k-means contains fewer operations than TRACLUS, and another one is that the data packing was implemented for k-means. This can also be seen in the complexity evaluation of PHE-based pp-TRACLUS shown in Table 7.1 in terms of communication and computational costs and note that n is the dataset size.

	Data Owner	**Cloud Server**	**EDServer**
Encryption	$8 \cdot n$		$4 \cdot (n^2 - n)$
Decryption			$8 \cdot (n^2 - n)$
Multiplication		$(8 + 12 + 12) \cdot (n^2 - n)$	
Exponentiation		$(8 + 4) \cdot (n^2 - n)$	
RNG[3]		$8 \cdot (n^2 - n)$	
Communication	1	$4 \cdot (n^2 - n)$	$4 \cdot (n^2 - n)$

Table 7.1 – Computational and communication complexity of pp-TRACLUS

Summary

In this section, we have presented our PHE-based pp-TRACLUS solution. We propose to approximate the TRACLUS distance metric with the use of Squared Euclidean Distance

to be compatible with the Paillier encryption scheme. Moreover, since Paillier enables additions and scalar multiplications, we present a secure multiplication protocol to perform multiplications and thus the Euclidean distance computations. In order to show the performance of our solution, we employ two datasets and compare our results with the privacy-preserving k-means.

7.4.3 2PC-based pp-TRACLUS: Description

This section discusses the limitation of the PHE-based pp-TRACLUS and investigates the approximated distance measure d_{pptrac} once more. Further, we provide our two-server aided and 2PC-based solution, which can be considered as a performance improvement on the data owner(s) side in the solution presented in Section 5.4.

The previously presented solution for pp-TRACLUS based on PHE (see Section 7.4.2) leaks some information that the cloud server learns the number of line segments and the values of $MinLns$ and ϵ. Moreover, although EDServer, which helps the cloud server perform the Euclidean distance and comparison computations, can extract no information about the coordinates of the line segments since the data randomisation/masking is performed, EDServer can access to ϵ^2 and $MinLns$ and can get the information about cluster sizes and which indices in the array belong to a cluster when it is analysing which line segment belongs to a neighbourhood. One can improve this solution by employing some permutations and some optimisation methods such as data packing in terms of security and performance aspects. With implementing permutations, EDServer cannot infer any information such as the cluster sizes.

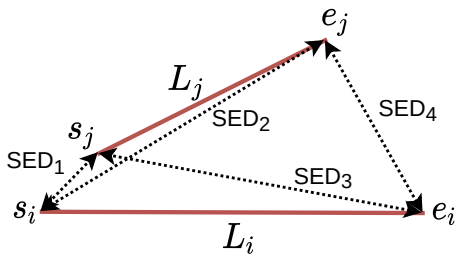


Figure 7.6 – Approximated distance measure d_{pptrac}

As described in Section 2.2.3 and illustrated in Figure 2.3, the computation of the TRACLUS distance metric *dist* involves some complex operations. As in Section 5, in order to ensure data privacy, to execute the clustering phase efficiently, and to obtain a high quality evaluation of TRACLUS, we approximate this distance metric *dist* to the combination of the Squared Euclidean Distance (SED), namely d_{pptrac} as depicted in Figure 7.6. This approximated distance measure is different than the approximated distance measure in Section 7.4.2 and consists of four SEDs since we restudy the quality evaluation of TRACLUS: We have conducted several simulations and discovered that the combination of four SEDs results in a high quality evaluation as the plaintext TRACLUS distance metric. Thus, the simplified distance measure for given two line segments L_i and L_j defined with starting points s_i and s_j and ending points e_i and e_j is as follows:

$$d_{pptrac}(L_i, L_j) = \mathsf{SED}(s_i, s_j) + \mathsf{SED}(s_i, e_j) + \mathsf{SED}(e_i, s_j) + \mathsf{SED}(e_i, e_j) \tag{7.4}$$

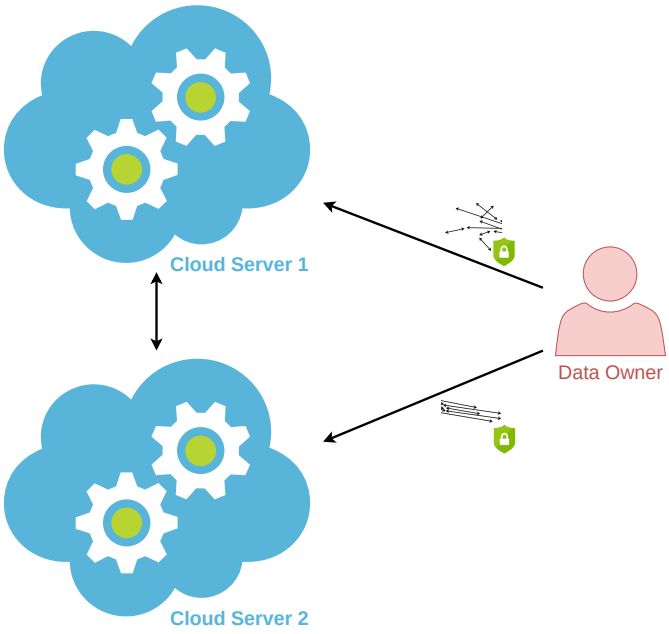


Figure 7.7 – Clustering in the two-server scenario

Therefore, this section proposes a 2PC-based solution to handle the limitation of the PHE-based solution. Similarly, employing two non-clouding servers in this solution is to have an efficient solution that incurs very little work for the data owner. For the 2PC-based pp-TRACLUS, we consider the following scenario illustrated in Figure 7.7: A data owner DO holds a dataset of line segments and delegates the trajectory clustering operations to two untrusted (semi-honest), non-colluding cloud servers CS1 and CS2. The reason for involving two non-colluding cloud servers instead of one is to move heavy computations between DO and CS to the computations between two cloud servers, namely CS1 and CS2. We assume that the data owner has already run the partitioning phase over its dataset D and would like to outsource the clustering phase over D to CS1 and CS2. DO first creates two secret shares using AS . Share (see Section 3.2.3) of the line segments, ϵ^2, and $MinLns$, and sends them to CS1 and CS2. Hence, non-colluding CS1 and CS2 would interactively execute each operation of the TRACLUS clustering phase under privacy preservation: They compute the distance matrix using the shares of line segments and ϵ^2 and then compute the clusters using the shares of this distance matrix and $MinLns$. Finally, each CS obtains one share of the pp-TRACLUS clusters' result. The cloud servers send their share to the data owner who adds them to get the final result.

Security Evaluation

In pp-TRACLUS based on 2PC, the data owner(s) and the two cloud servers are assumed to be semi-honest: they have to run the protocol steps correctly, yet they may try to collect exchanged information and further may use these information to obtain any information about the input and output of pp-TRACLUS. pp-TRACLUS is secure and ensures data privacy thanks to the use of 2PC.

Performance Evaluation

We evaluate the performance of the 2PC-based pp-TRACLUS related to a real-life use case scenario in which a data owner DO, namely a telecommunication provider, collects human movements trajectories for the analysis purposes related to the containment of COVID-19 and have already run the partitioned phase of TRACLUS. Further, DO owns the dataset of line segments and wishes to learn similarities or some pattern among them by employing two non-colluding semi-honest cloud servers CS1 and CS2. Due to the nature of the dataset (i.e., privacy-sensitive), and thus before outsourcing, DO splits all line segments into two shares and sends one share to CS1 and one share to CS2. Once CS1 and CS2 complete the protocol run, each holds the share of the result, which will be sent to DO. Finally, DO obtains the clustering result.

To implement our solution, we use the ABY framework [42] written in C++, and the security level is set to 128-bit. All experimental setup and the datasets are the same as the 2PC-based pp-TRACLUS containing players of DO and CS in Section 5.4.3 except we employ two separate cloud servers, each equipped with 2.30 GHz Intel Xeon Gold 6140 processor with Ubuntu 20.04 LTS and 64 GB RAM.

The pp-TRACLUS performance results are depicted in Table 7.2. These results correspond to the average from the execution of 10 individual simulations. We observe that a pp-TRACLUS instance on the Travel dataset takes 16.66 mins for 400 line segments utilising arithmetic sharing for the distance computation and Boolean sharing for the clusters whereas using only Boolean sharing, it takes 17.53 mins for 400 line segments.

Table 7.2 – Timing result for pp-TRACLUS on the Travel dataset with $\epsilon^2 = 13.5 \times 10^9$ and $MinLns = 3$

pp-TRACLUS	# of Line Segments	Time (mins)	Memory (GB)
Arithmetic & Boolean Sharing	100	1.25	2
	400	16.66	11
	1000	53.20	62
Boolean Sharing	100	1.10	2
	400	17.53	11
	1000	113.26	62

To summarise, our simplified and approximated distance measure d_{pptrac} containing the use of two-server lowers the DO's workload and securely clusters line segments without sacrificing data privacy and leaking any information about the clusters.

This particular dataset (illustrating persons' travel patterns) may need more precision (i.e., more clusters). In other words, there might be more than one route between two locations A and B, and these routes should not have been grouped into one: pp-TRACLUS can group all line segments in one cluster because of the *ExpandCluster* method in Algorithm 5. We propose to reduce the number of *ExpandCluster* to 1 in order to take only the first-level neighbours into account and to obtain a larger variety of clusters (containing trajectories that reach B by passing through different locations $L_i, 0 \leq i$) and call this adaption pp-TRACLUS′. We run two experiments for pp-TRACLUS′, namely Experiment 1 and 2 that have different ϵ^2 values. Our results show that the number of clusters increases with good quality evaluation by SC, SC_{noise}, and DBCV when comparing TRACLUS and pp-TRACLUS.

Table 7.3 – Results of the clustering quality assessment for TRACLUS, pp-TRACLUS, and pp-TRACLUS′ on the Travel dataset. The best results are marked in bold.

	Experiment 1		
	TRACLUS	**pp-TRACLUS**	**pp-TRACLUS′**
(ϵ^2, $MinLns$)	$(4200, 3)$	$(450 \times 10^6, 3)$	$(450 \times 10^6, 3)$
# of Clusters	1	2	48
Noise	796	13092	13506
SC	N/A	0.83	**0.98**
SC_{noise}	N/A	0.56	**0.65**
DBCV	N/A	**0.67**	0.37

Table 7.4 – Results of Experiment 2 of the clustering quality assessment for TRACLUS, pp-TRACLUS, and pp-TRACLUS′ on the Travel dataset. The best results are marked in bold.

	Experiment 2		
	TRACLUS	**pp-TRACLUS**	**pp-TRACLUS′**
(ϵ^2, $MinLns$)	$(47 \times 10^3, 3)$	$(13.5 \times 10^9, 3)$	$(13.5 \times 10^9, 3)$
# of Clusters	1	1	2
Noise	0	359	361
SC	N/A	N/A	**0.82**
SC_{noise}	N/A	N/A	**0.81**
DBCV	N/A	N/A	**0.98**

We also evaluate the timing results for pp-TRACLUS′ shown in Table 7.5. These results correspond to the average from the execution of 10 individual simulations. The clustering on the Travel dataset takes 3.68 mins for 400 line segments utilising arithmetic sharing for the distance computation and Boolean sharing for the clusters whereas using only Boolean sharing for both computations, 400 line segments are clustered in 13.56 mins.

Our simplified and approximated distance measure d_{pptrac} creates a larger number

Table 7.5 – Timing result for pp-TRACLUS′ on the Travel dataset with $\epsilon^2 = 13.5 \times 10^9$ and $MinLns = 3$

pp-TRACLUS′	# of Line Segments	Time (mins)
Arithmetic & Boolean Sharing	100	0.43
	400	3.68
	1000	25.23
Boolean Sharing	100	0.66
	400	13.56
	1000	87.40

of clusters and marks more elements as outliers. Nevertheless, our clustering quality evaluation gives even better results for the Travel dataset showing that its quality evaluation is comparable to the original tripartite distance metric of TRACLUS.

7.5 Conclusion of privacy-preserving clustering

In this chapter, we have introduced new privacy-preserving trajectory clustering solutions, namely PHE-based pp-TRACLUS and 2PC-based pp-TRACLUS, that allow two non-colluding cloud servers to efficiently cluster a dataset coming from a data owner and obtain a good clustering quality evaluation without scarifying the privacy of the underlying data.

In these solutions, we implement the privacy-by-design approach and consider two cases: (i) minimising the cryptographic technique incompatibilities; and (ii) providing private trajectory clustering.

- PHE-based pp-TRACLUS proposes an approximate for the TRACLUS distance metric with the use of Squared Euclidean Distance which is efficiently performed by the Paillier encryption scheme. Moreover, we present a secure multiplication protocol to perform multiplications and thus the Euclidean distance computations. Moreover, we compare our performance results with the privacy-preserving k-means that uses Squared Euclidean Distance.

- 2PC-based pp-TRACLUS also employs a similar distance measure as the PHE-based pp-TRACLUS; however, the former clusters a given dataset without leaking any information. Moreover, we evaluate the performance and also the clustering quality: Our solution, pp-TRACLUS with d_{pptrac}, enables efficient privacy preservation with a good level of clustering quality.

Chapter 8

Privacy-preserving Data Aggregation

The earth has music for those who listen.

George Santayana

In this chapter, we study the privacy challenges raised by data aggregation when integrating cryptographic techniques. Then, we review the state-of-the-art solutions and further present our solution consisting of two cloud servers for privacy-preserving data aggregation based on multi-key fully homomorphic encryption, multi-key TFHE, and threshold fully homomorphic encryption.

8.1 Introduction

According to The Economist[1], data is stated as the new, most valuable resource of the world. Data-driven companies are always in competition to collect/process more and more information about their clients (or users) and further utilise them to improve their services. However, companies being abundant of such data need the data to be gathered/expressed in a summary form. Thanks to data aggregation which consists of collecting data and performing some statistical analysis such as sum, average, etc., these companies can make correct decisions and improve their services quickly, and this thus increasingly makes such aggregate data essential and valuable for companies since the aggregate data is ready to bring answers to analytical questions for groups of people or helps companies achieve insights about their business analysis. Nevertheless, the sensitive nature of the data raises serious privacy and legal concerns. Therefore,

[1] https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data

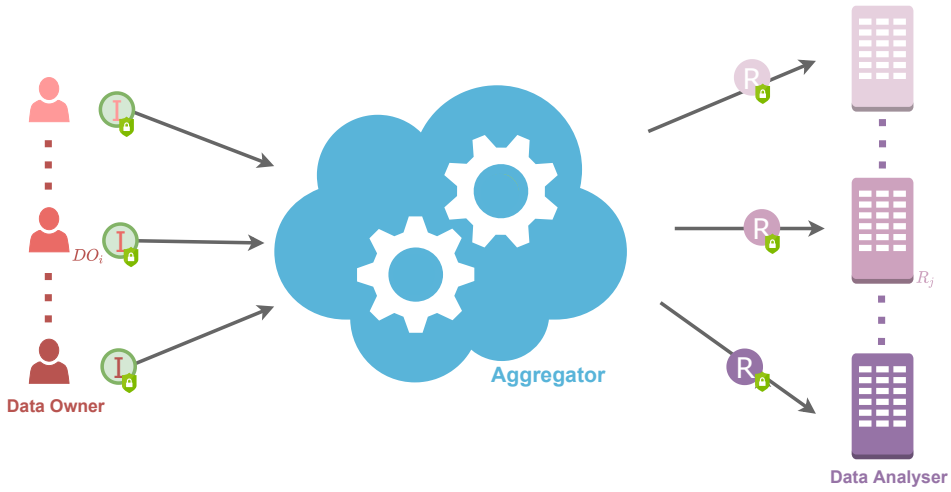


Figure 8.1 – Data aggregation setting

companies require the implementation of cryptographic techniques which, on the one hand, should ensure data privacy and help comply with privacy regulations such as the General Data Protection Regulations (GDPR) [2] and on the other hand, should be compatible with the underlying data processing/data aggregation operations.

Furthermore, companies may need domain-specific expertise and/or computational resources to perform the aggregation operations over protected data. Hence, they usually rely on the existence of third party cloud servers for such analytics operations. In this chapter, we name these cloud servers *Aggregators*. These can be considered as one platform which offers aggregation operations for multiple stakeholders and over multiple datasets collected from a large number of clients. This platform can help companies boost their quality of service. With the existence of this new third-party *Aggregator*, the implementation of cryptographic techniques becomes even more essential.

A privacy-preserving data aggregation protocol is defined as a protocol where an *Aggregator* collects data from a large number of clients that we name *Data Owners* in a privacy-preserving manner and obtains some aggregate information about these collected data without leaking the individual data values. The aggregation operation usually consists of the sum of the collected data. Moreover, there are some *Data Analyser* that one Aggregator may serve with the data collected from several Data Owners. In this chapter, we study the privacy challenges raised by this setting as illustrated in Figure 8.1 and propose PRIDA, a privacy-preserving data aggregation solution that, on the demand of some stakeholders, allows untrusted third-party *Aggregators* to collect private data from a large number of clients and perform some aggregation operation over these data. PRIDA ensures that the aggregate result is only accessible to the actual stakeholder who queried the aggregation operation.

8.2 Privacy vs. Data Aggregation

This section focuses on data aggregation and its operations to identify the privacy requirements/goals to design the privacy protection variant of it.

Privacy-preserving data aggregation solutions enable an untrusted cloud server to collect data, gather or express the privacy-sensitive data in a summary form and perform some statistical analysis such as sum, average, etc., over them. Companies, which are interested in the aggregated result, utilise the services of Machine Learning as a Service consisting of domain-specifically expert and/or computationally resourced cloud server(s) (namely Aggregator(s)) to perform the aggregation operations over protected data. With the existence of these outsourced services, the implementation of cryptographic techniques becomes even more essential due to the data protection regulations [2, 3].

By definition, a privacy-preserving data aggregation protocol should ensure *input privacy*: Each input to the aggregation operation that is collected from Data Owners (DOs) should remain confidential to all parties except the actual DO. Furthermore, because the newly proposed setting introduces multiple Data Analysers (DAs), given an instance of the protocol, only authorised DAs should be able to have access to the actual aggregate result. This can be defined as the need for *output privacy*.

Additionally, since the Aggregator (Agg) is serving multiple DAs interested in receiving aggregate information, DOs can easily lose the control of the use of their data. Indeed, DOs cannot easily *control* whether they have or not participated to an aggregation operation requested by a certain DA. We believe that this problem has not been studied in existing solutions since these consider a unique DA or Agg who is inherently authorised to receive the result. Indeed, in prior solutions, if DO does not want to contribute to the statistics, it simply does not send its input to Agg. This is no more possible when there exists multiple DAs.

Finally, we believe that, in this new setting, DOs may also wish to remain *anonymous* to DAs: DAs should not even identify which DO participated to the actual aggregation operation. Existing works do not discuss this problem mainly because Agg knows all the participating DOs.

8.3 Prior Work

This section overviews existing privacy-preserving data aggregation solutions based on the cryptographic techniques.

8.3.1 DP (and HE)-based solutions

Rastogi and Nath [187] employ a distributed Laplace perturbation algorithm to add noise in a distributed way to ensure differential privacy (DP) of inputs. Also, authors consider potential collusions between some Data Owner (DO) and the Aggregator or collusions between DOs. Thanks to DP, when collusions exit, colluding parties cannot disclose the input data. However, [187] cannot be enough to meet the privacy requirements in the many-to-many setting. Shi et al. [188], Kursawe et al. [189], and Chan et al. [190] utilise

DP and Decisional Diffie-Hellman (DDH) for the input privacy against the semi-honest Aggregator. [191, 192] generalise the study of [188] by using the Paillier cryptosystem [9] and Aggregator-oblivious encryption scheme, respectively. Note that these solutions [188–192] need a trusted dealer for key distribution among DOs, and there is no other party, namely a Data Analyser (DA), other than the Aggregator acting as DA. When the many-to-many setting is considered, the data aggregation for another DA needs new keys to be generated. [193] is an extension of [188, 191] since these solutions require a trusted key dealer and key updates, and it enables DOs to generate their self-generated keys, which a semi-trusted collector obtains. Later this collector constructs the key for the Aggregator. Yet, [193] is not sufficient to be a solution for the challenges in the many-to-many setting. Bilogrevic et al. [194] employ an encryption scheme and DP. The aggregate result is open to Aggregator, and Aggregator can sell this result to some Customer(s) as Data Analysers. Although the solution by [194] seems perfect to satisfy the privacy requirements in the many-to-many setting, the result privacy is neglected, and also, DOs do not control DAs. Moreover, PATE [195], [196], UnLynx [197], Drynx [198], and SPINDLE [199] propose private learning over distributed datasets. Differently, [200] and [201] present a secure shuffle model of DP whereby DOs send their private data through a shuffler, which ensures the Aggregator not to identify which data sent by which DOs.

8.3.2 HE-based solutions

Erkin [202] proposes to combine Chinese Remainder Theorem with the modified Paillier cryptosystem. DOs are split into hierarchical groups using the public key of Utility Provider as DA, and DOs in these groups are required to communicate with each other for generating a secret key. The computation of aggregate data is performed over these groups' encrypted data by Aggregator. Then, DA receives the aggregate result. When adopting this solution to the many-to-many setting, DOs cannot easily choose DAs, and if Aggregator and DA collude, the input privacy is eliminated. Leontiadis et al. [203] propose PUDA (needs a trusted dealer to generate keys) whereby DOs send their private using DDH to Aggregator who computes and learns the aggregate result. Then, Aggregator forwards this result to some DA. While considering the many-to-many setting, the output privacy is eliminated. [204] uses DHH-based private set intersection (PSI) with the Paillier scheme and consists of two parties: one obtains PSI, and the other receives the sum over the PSI result. This solution cannot easily be modified since the many-to-many setting poses a large number of parties, and PSI is not the scope of PRIDA.

8.3.3 MPC/2PC-based solutions

Many DOs of SEPIA [205] export their input data to the SEPIA input peers, and these peers split inputs into shares using Shamir's secret sharing [206] and send to the SEPIA privacy peers as Aggregators. The result is sent back to DOs. However, in SEPIA, the input is not private only to its owner. In Prio [207], many DOs use secret sharing to split their data and send these shares to a set of servers as Aggregators. These Aggregators compute data analytics over them and obtain the aggregate statistics, which is not

confidential only to a (or many) DA(s). Moreover, Bonawitz et al. [208] utilise Shamir's secret sharing whereby many DOs generate various random masks for their private inputs to add/subtract these random numbers to/from their input and send their secret keys shares to Aggregator. When all the inputs are brought together, Aggregator can learn the addition result. Since the need for collaboration between DOs is not wanted in the many-to-many setting, [208] cannot be easily adapted. Tsaloli et al. [209] propose a verifiable Shamir's secret sharing based aggregation. Helen [210] is also an MPC-based platform for collaborative learning. Additionally, SAFER [211] employs MPC to provide a secure aggregation protocol performed by several Aggregators. F2ED-Learning [212] uses a mean estimator, FilterL2 and secure aggregation (on top of [208]) for federated learning. In [212], all DOs are split into many small groups that perform secure aggregation with the help of a centralised Aggregator. Further, FilterL2 is used over the aggregated local updates from different groups. Furthermore, FLGUARD [213] proposes secure 2PC to guarantee input privacy. Prio+ [214] is a work on top of Prio [207], but DOs use Boolean secret sharing to ensure some set of Aggregators for convincing their inputs used in a correct form. Lastly, SAFELearn [215] is a federated learning solution based on MPC or based on FHE. A recent study proposed by Karakoç et al. [216] use oblivious programmable pseudo-random function. Nevertheless, with these MPC-based solutions, DOs cannot choose which DAs can receive their input used to compute data aggregation, and the anonymity of DOs cannot be provided.

8.3.4 Hybrid solutions

Some hybrid solutions [217, 218] use the Paillier encryption and additive secret sharing. Also, [219] utilise pairwise non-interactive key exchange (NIKE), outsource this NIKE to the non-colluding Aggregators, and with a mask-then-encrypt technique, authors provide the input privacy. Nevertheless, these solutions do not consider the existence of multiple DAs and they cannot be easily extend to the challenges described in Section 8.4.1.

To summarise, most of the state-of-the-art solutions do not consider multiple DAs, and do not provide flexibility for DOs, who can easily choose some DAs to have access to the aggregate result. Therefore, most of the existing solutions can be considered as an "'all-or-nothing"' whereby as soon as the DO participates to the aggregation, all DAs can have access to the aggregate result. Additionally, only one work [187] studies the effect of collusions between parties. Finally, the problem of DO anonymity is not tackled in many solutions. Thus, we can conclude that by combining the use multi-party fully homomorphic encryption (MP-FHE that contains multi-key FHE and threshold FHE) and 2PC and introducing two non-colluding Aggregators, PRIDA succeeds in supporting multiple DAs while ensuring input and output privacy, allowing DOs to have some control on their data and keep their anonymity.

8.4 PRIDA: PRIvacy-preserving data aggregation with multiple Data Analysers

In this section, we propose PRIDA [220], a privacy-preserving data aggregation solution with multiple data analysers. This work is under submission.

In this work, we consider a more realistic (and more generic) scenario where there are more than one *Data Analyser* that one Aggregator may serve with the data collected from several Data Owners.

8.4.1 Problem Statement

In this section, according to the identified privacy requirements/goals raised by the extended setting in Section 8.2, we revise the threat model accordingly.

As *input privacy* is the traditional and essential privacy requirement for a privacy-preserving data aggregation solution, the state-of-the-art satisfies this requirement. However, the new setting introducing many Data Analysers raises new challenges such as the need for *output privacy*, the lack of *control of DOs over DAs* and *DO anonymity*. Therefore, this new setting is needed to revisit the threat model of a privacy-preserving data aggregation protocol involving multiple DAs. The introduction of multiple DAs increases the risk of potential corruptions and collusions among parties which could seriously endanger input and output privacy guarantees.

Similar to the majority of existing studies, we assume a semi-honest security model where all parties have to follow the protocol steps correctly. Still, these can act curiously to infer some information from the exchanged data. The potential adversaries in this threat model are enumerated as follows:

- An *external adversary* who does not participate to the protocol may try to discover information about inputs of DOs and/or results (or outputs) of DAs.
- The semi-honest *Data Owner* (DO) may try to learn information about the data aggregation result(s) and other DOs' inputs. Moreover, note that DOs should not discover the other DOs who contribute the aggregate information.
- The *Aggregator* (Agg) may wish to discover input data of DOs and DAs' aggregate results.
- *Data Analyser* DA may try to discover information about inputs of DOs and other DAs' aggregate results. DA may also try to discover the identity of any DO.

As previously mentioned, the existence of multiple DAs can imply more power to the adversaries with potential collusions. For example, if some DA and Agg collude, they can discover the input of some DOs. Also, the output privacy guarantee can be in danger since the result of other DA(s) can be disclosed by these two parties when they collude. Therefore, the threat model should also consider the collusions between parties:

- When *curious DO and Agg collude*, the DO should not learn any information about the data aggregation result(s) and other DOs' inputs.

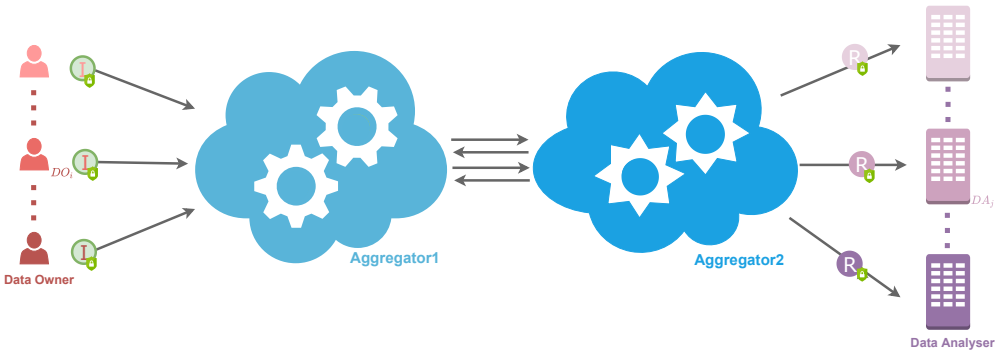


Figure 8.2 – PRIDA - Players

- If *DA and Agg collude*, DA should not learn anything about inputs of DOs and other DAs' aggregation results.
- Finally, Agg should not discover the input of DOs and DAs' results even if *Agg colludes with DA or DO*.

In order to cope with the challenges identified in the previous section, namely (i) the possible collusions between parties; (ii) the lack of control of DOs over DAs; and (iii) the anonymity of DOs in addition to the traditional privacy requirements that are input and output privacy, we propose a new solution named PRIDA that firstly introduces two non-colluding Aggregators instead of one. This new setting protects against collusions among different parties. Furthermore, to ensure both input and output privacy and enable DO to have some control over DAs, we propose to combine the use of MP-FHE whereby one can perform operations over data encrypted under multiple keys and secure two-party computation whereby two parties jointly perform a function over their private inputs without disclosing them. The 2PC protocol is implemented between the two Aggregators and mainly ensures input and output privacy against Aggregators. On the other hand, the use of MP-FHE enables DO to encrypt the data with multiple keys corresponding to the Aggregators and DA, helps DOs have some control over DAs. We believe that each cryptographic technique alone is not sufficient to achieve all privacy goals identified for PRIDA: Indeed, if 2PC is used alone, unauthorised DAs can receive some aggregate result when colluding with one Agg; on the other hand, if MP-FHE is used alone, the inputs of DOs can be revealed, and DOs cannot easily control DAs when there is collusion between Agg and DA.

Furthermore, in order to protect the anonymity of Data Owners, before the actual privacy-preserving aggregation operation, we propose a preliminary privacy-preserving counting scheme that ensures that aggregation takes place only if some threshold number is reached.

To summarise, PRIDA involves the following three parties in Figure 8.2:

- *Data Owner* DO owns some confidential input data and outsources this confidential data to the Aggregators once its data is first encrypted with MP-FHE and further

secret shared. DO also secretly defines which Data Analyser can have access to the aggregate result involving its input.

- *Data Analyser* DA obtains the data aggregation result over the inputs from many DOs in cleartext if authorised using the decryption algorithm of MP-FHE.
- The two *Aggregators* (Agg1 and Agg2) are two non-colluding cloud servers which collect the encrypted data from many DOs, perform the data aggregation requested by DAs, and send the results to the authorised DA.

8.4.2 PRIDA: Detailed description

In this section, we present our solution named PRIDA, which combines the use of 2PC with MP-FHE schemes in order to ensure that Data Owners (DOs) can choose which Data Analyser (DA) can have access to the aggregate result involving their input, and, at the same time, in order to protect against potential collusions among parties. As previously mentioned, the only assumption we make is that the two Aggregators do not collude.

PRIDA is defined in five phases: (i) *the setup phase* in which the keying material is generated; (ii) *the data protection phase* whereby DO actually decides which DA can have access to the aggregate result for which DO has participated and encrypts and secret shared the relevant data accordingly; (iii) *the preliminary counting phase* in which the two Aggregators find out which DA can receive the aggregate result is by counting number of authorising DOs and verifying if this number exceeds a pre-defined threshold; (iv) *the aggregation phase* which consists of the Aggregators jointly aggregating the data collected from different DOs; and finally (v) *the decryption phase* whereby an authorised DA decrypts the data aggregation result together with the two Aggregators.

We propose three versions of PRIDA which differ with respect to the underlying MP-FHE scheme:

- **PRIDA v1** makes use of the asymmetric MK-FHE [87] (see Section 8.4.2). Each DO replicates its DAta as many times as the number of authorised DAs and encrypts each instance with three public keys: The public keys of the two Aggregators and the public key of the actual DA.
- **PRIDA v2** is based on the symmetric MK-FHE, namely MK-TFHE [88] (see Section 8.4.2). In this case, DOs need to establish pairwise symmetric keys with each Aggregator.
- **PRIDA v3** is based on Th-FHE [90, 91] (see Section 8.4.2) and consists of each DO encrypting the data with one common public key that is collaboratively generated by one DA and the two Aggregators. The decryption involves the three secret keys of each party corresponding to this common public key.

As previously mentioned, during the *setup phase*, parties generate their keying material according to the underlying multi-party homomorphic encryption scheme. In the second

phase, each DO_i decides which DA is authorised to have access to the aggregate result in which its input is involved. To this end, DO_i first defines a binary *choice vector* $\mathbf{cv}_i$ where each element is mapped to a particular DA and the value corresponds to the authorisation decision (0 if DA is not authorised and 1 if authorised). Each DO also defines a *data vector* $\mathbf{dv}_i$ of dimension m (same size with $\mathbf{cv}$) where m is the total number of DAs. If the actual DA is authorised, then the corresponding element is set to the input of DO. Otherwise, DO generates a random number r which is set to the corresponding element of vector $\mathbf{dv}$. Finally, DO randomly generates two arithmetic secret shares for $\mathbf{cv}$ and $\mathbf{dv}$ and further encrypts the two shares of $\mathbf{dv}$. Each share is then sent to the corresponding Aggregator (Agg$k$. $k = 1$ or 2). Then, in the *preliminary counting phase* Agg1 and Agg2 jointly add the choice vectors received from each DO in 2PC and further obtain the resulting number of DOs per DA, cv_{total_j} without discovering which DO authorised which DA_j. If the number of DOs authorising a particular DA_j is greater than the pre-defined threshold t, then the two Aggregators can start the actual *aggregation phase* described in Algorithm 6: For an authorised DA_j, Aggregators compute $\mathrm{cv}_{ij} \cdot [\mathrm{dv}_{ij}]$ for each DO_i in 2PC and further compute the sum of these intermediate values for all DO_i to find the aggregate result $[s_j]$ for the authorised DA_j. Finally, during the *decryption phase*, Aggregators partially decrypt the aggregate result which is further sent to the corresponding authorised DA who in its turn partially decrypts obtains the plaintext result s_j.

Algorithm 6 Aggregation phase executed by Agg1 and Agg2

for $i = 1$ *to* n **do**
- Compute $[\langle \boldsymbol{\epsilon}_i \rangle_k] \leftarrow \mathrm{Eval}(+, ([\langle \mathbf{dv}_i \rangle_k], \langle \boldsymbol{\alpha}_i \rangle_k))$.
- Calculate $\langle \boldsymbol{\delta}_i \rangle_k \leftarrow \langle \mathbf{cv}_i \rangle_k + \langle \boldsymbol{\beta}_i \rangle_k$.
- Agg1, Agg2: Exchange $[\langle \boldsymbol{\epsilon}_i \rangle_k]$ and $\langle \boldsymbol{\delta}_i \rangle_k$ to compute $[\boldsymbol{\epsilon}_i]$ and $\boldsymbol{\delta}_i$.
- Use Eval to element-wise multiply $[\boldsymbol{\epsilon}_i]$ and $\langle \mathbf{cv}_i \rangle_k$.
- Employ Eval to element-wise multiplies $\boldsymbol{\delta}_i$ and $[\langle \mathbf{dv}_i \rangle_k]$.
- Add $\langle \boldsymbol{\gamma}_i \rangle_k$, $[\boldsymbol{\epsilon}_i \star \langle \mathbf{cv}_i \rangle_k]$, and $[\boldsymbol{\delta}_i \star \langle \mathbf{dv}_i \rangle_k]$ by making use of Eval.
- Agg1: Send $[\langle \boldsymbol{\gamma}_i \rangle_k + \boldsymbol{\epsilon}_i \star \langle \mathbf{cv}_i \rangle_k + \boldsymbol{\delta}_i \star \langle \mathbf{dv}_i \rangle_k]$ to Agg2.
- Agg2: Perform $[\langle \boldsymbol{\gamma}_i \rangle + \boldsymbol{\epsilon}_i \star \langle \mathbf{cv}_i \rangle + \boldsymbol{\delta}_i \star \langle \mathbf{dv}_i \rangle] \leftarrow \mathrm{Eval}(+, ([\langle \boldsymbol{\gamma}_i \rangle_1 + \boldsymbol{\epsilon}_i \star \langle \mathbf{cv}_i \rangle_1 + \boldsymbol{\delta}_i \star \langle \mathbf{dv}_i \rangle_1], [\langle \boldsymbol{\gamma}_i \rangle_2 + \boldsymbol{\epsilon}_i \star \langle \mathbf{cv}_i \rangle_2 + \boldsymbol{\delta}_i \star \langle \mathbf{dv}_i \rangle_2]))$.
- Agg2: Compute $[\boldsymbol{\epsilon}_i \star \boldsymbol{\delta}_i] \leftarrow \mathrm{Eval}(\star, ([\boldsymbol{\epsilon}_i], \boldsymbol{\delta}_i))$.
- Agg2: Calculate $[\mathbf{dv}_i \star \mathbf{cv}_i] \leftarrow \mathrm{Eval}(-, ([\boldsymbol{\gamma}_i + \boldsymbol{\epsilon}_i \star \mathbf{cv}_i + \boldsymbol{\delta}_i \star \mathbf{dv}_i], [\boldsymbol{\epsilon}_i \star \boldsymbol{\delta}_i]))$.

Agg2: Call Eval to add all $[d_{ij} \cdot c_{ij}]$ and finds aggregate result $[s_j]$ for authorised DA_j.

We now present the details of PRIDA depicted in Protocol 5, Protocol 6, and Protocol 7 which is based on 2PC and MK-FHE, MK-TFHE, and Th-FHE. In the next sections, we will only highlight the main differences between each instantiation.

PRIDA v1: PRIDA with MK-FHE

PRIDA v1 presented in Protocol 5 combines secure two-party computation (2PC) and asymmetric multi-key fully homomorphic encryption (MK-FHE) [87].

During the *data protection phase* of Protocol 5, each DO_i first needs to secret share

its data $\mathbf{dv}_i$ and encrypt each share with three public keys: those of Agg1, Agg2, and the actual DA_j. Since the original MK-FHE only involves one public key, we propose the following transformation (Step (b) to (f) in Protocol 5): DO_i once again secret shares $\langle \mathbf{dv}_i \rangle_k$ into three shares and encrypts each share with one of the three public keys; then, MK-FHE . Pre-process is called for each of these encrypted shares and finally these three values are summed using MK-FHE . Eval. The obtained result correspond to the encryption of $\langle \mathbf{dv}_i \rangle_k$ with three keys, as expected.

PRIDA v2: PRIDA with MK-TFHE

This section shows that PRIDA also supports a symmetric multi-key FHE. As previously mentioned in Section 3.5.2, a symmetric multi-key FHE, namely MK-TFHE, is constructed as an extension of TFHE [63]. The main challenge to build on MK-TFHE relies on the generation and distribution pairwise symmetric keys. The large number of parties (DOs and DAs) results in a large number of pairwise keys. We therefore address this problem by transforming ciphertexts encrypted with individual key sets to ciphertexts encrypted with common *group* keys. More specifically, each DA_j shares one common key k_j with all DOs and each DO establishes one pairwise key with each Aggregator; When Agg1 and Agg2 receive individual ciphertexts encrypted with different sets of keys, they partially decrypt them with the keys that they know and re-encrypt them with their unique key only known by themselves. We introduce this new algorithm in Algorithm 7 as MK-TFHE . Post-process. Thanks to this new algorithm, the two Aggregators do not need to execute MK-TFHE . Pre-process as double as the number of DOs.

The specification of PRIDA v2, executing MK-TFHE . Post-process is provided in Protocol 6, and the main difference with Protocol 5 resulting from the use of MK-TFHE instead of MK-FHE is highlighted with the blue colour.

PRIDA v3: PRIDA with Th-FHE

PRIDA v3 is constructed by using 2PC and threshold fully homomorphic encryption (Th-FHE). As previously mentioned, Th-FHE requires each DA and the two Aggregators to jointly generate one unique public key. Therefore, in the *setup phase*, they employ Th-FHE . KeyGen to generate a public key for the use by DOs. The details are shown in Protocol 7, and the different steps of Protocol 7 are coloured with blue when compared to Protocol 5.

Protocol 5 PRIDA based on asymmetric MK-FHE

Inputs. DO_i, $i \in \{1, \ldots, n\}$, inputs a choice vector $\mathbf{cv}_i$ and a data vector $\mathbf{dv}_i$ of size m. Also, a pre-defined threshold t and the public parameters pp are published.
Output. If $\mathrm{cv}_{total_j} \geq t$, DA_j obtains the aggregate result s_j, $j \in \{1, \ldots, m\}$. Otherwise, DA_j obtains nothing.
Protocol steps:

1. *Setup executed by DA_j, Agg1, and Agg2.*
   (a) Generate $(\mathsf{sk}_p, \mathsf{pk}_p) \leftarrow \text{MK-FHE}.\mathrm{KeyGen}(\mathsf{pp})$ where $p =$DA$_j$, Agg1, and Agg2.
2. *Data protection executed by DO_i.*
   (a) Generate $(\langle \mathbf{cv}_i \rangle_1, \langle \mathbf{cv}_i \rangle_2) \leftarrow \mathrm{AS}.\mathrm{Share}(2, \mathbf{cv}_i)$.
   (b) Generate $(\langle \mathbf{dv}_i \rangle_1, \langle \mathbf{dv}_i \rangle_2) \leftarrow \mathrm{AS}.\mathrm{Share}(2, \mathbf{dv}_i)$.
   (c) Generate $\left(\langle \mathrm{dv}_{ij} \rangle_{k,\mathrm{DA}_j}, \langle \mathrm{dv}_{ij} \rangle_{k,\mathrm{Agg1}}, \langle \mathrm{dv}_{ij} \rangle_{k,\mathrm{Agg2}}\right) \leftarrow \mathrm{AS}.\mathrm{Share}(3, \langle \mathrm{dv}_{ij} \rangle_k)$ for $k = 1, 2$.
   (d) $[\langle \mathrm{dv}_{ij} \rangle_{k,p}] \leftarrow \text{MK-FHE}.\mathrm{Encrypt}(\langle \mathrm{dv}_{ij} \rangle_{k,p}, \mathsf{pk}_p)$ for $k = 1, 2$ and $p =$ DA$_j$, Agg1, Agg2.
   (e) Call MK-FHE . Pre-process with all $[\langle \mathrm{dv}_{ij} \rangle_{k,p}]$ for $T^* = \{id_{DA_j}, id_{Agg1}, id_{Agg2}\}$.
   (f) $[\langle \mathrm{dv}_{ij} \rangle_k] \leftarrow \text{MK-FHE}.\mathrm{Eval}(+, ([\langle \mathrm{dv}_{ij} \rangle_{k,1}], [\langle \mathrm{dv}_{ij} \rangle_{k,2}], [\langle \mathrm{dv}_{ij} \rangle_{k,3}]))$, $k = 1, 2$.
   (g) Generate random vectors $\boldsymbol{\alpha}_i$, $\boldsymbol{\beta}_i$ and compute $\boldsymbol{\gamma}_i$ such that $\boldsymbol{\gamma}_i = \boldsymbol{\alpha}_i \star \boldsymbol{\beta}_i$.
   (h) *Beaver's triplets.* Call AS . Share(2, .) for $\boldsymbol{\alpha}_i$, $\boldsymbol{\beta}_i$, $\boldsymbol{\gamma}_i$.
   (i) Send $[\langle \mathbf{dv}_i \rangle_k]$, $\langle \mathbf{cv}_i \rangle_k$, $\langle \boldsymbol{\alpha}_i \rangle_k$, $\langle \boldsymbol{\beta}_i \rangle_k$, and $\langle \boldsymbol{\gamma}_i \rangle_k$ to Agg$k$, $k = 1, 2$.
3. *Preliminary counting executed by Agg1 and Agg2.*
   (a) Obtain $\langle \mathbf{cv}_{total} \rangle_k$ such that $\sum \langle \mathbf{cv}_i \rangle_k$.
   (b) Agg1 and Agg2 exchange $\langle \mathbf{cv}_{total} \rangle_k$ to get $\mathbf{cv}_{total} = (\mathrm{cv}_{total_1}, \ldots, \mathrm{cv}_{total_j}, \ldots, \mathrm{cv}_{total_m})$.
   (c) DA_j is labelled as authorised if $\mathrm{cv}_{total_j} \geq t$.
4. *Aggregation executed by Agg1 and Agg2.*
   (a) Jointly compute $[\mathbf{s}] = \sum [\mathbf{dv}_i \star \mathbf{cv}_i]$ for each authorised DA_j. The details are provided in Algorithm 6.
5. *Decryption executed by Agg1, Agg2, and DA_j.*
   (a) Agg2: Send the aggregate result vector $[\mathbf{s}] = (\ldots, [s_j], \ldots)$ to Agg1.
   (b) Agg1,Agg2: Employ $\mu_k \leftarrow \text{MK-FHE}.\mathrm{PartialDecrypt}$ where $k = 1, 2$.
   (c) Agg1: Send μ_1 to Agg2.
   (d) Agg2: Send μ_1 and μ_2 to the authorised DA_j.
   (e) DA_j: Call $\mu_3 \leftarrow \text{MK-FHE}.\mathrm{PartialDecrypt}$.
   (f) DA_j: Run MK-FHE . Merge with μ_1, μ_2, μ_3 to find s_j.

Algorithm 7 MK-TFHE . Post-process

Input: $\bar{\text{ct}} = (b, a_1, \ldots, a_k), T = \{id_1, \ldots, id_l, \ldots, id_k\}$, old identity id_l, new identity id_l^*, and their respective secret keys

Output: ct^*, T^*

$b' \leftarrow b - \langle a_l, sk_{id_l} \rangle$. // n.b., noise is added in the next step

$b' \leftarrow b - \langle a_l, sk_{id_l} \rangle$. // n.b., noise is added in the next step

$(b^*, a_l^*) \leftarrow$ MK-TFHE . Encrypt$(b', sk_{id_l^*})$.

$\text{ct}^* = (b^*, a_1, \ldots, a_l^*, \ldots, a_k)$, $T^* = \{id_1, \ldots, id_l^*, \ldots, id_k\}$.

8.4.3 Security Evaluation

We analyse the security of PRIDA, taking the previously introduced threat model into account and show that PRIDA satisfies the privacy requirements defined in Section 8.4.1. We incrementally study the security analysis by considering each PRIDA player as an adversary and further investigating potential collusions. We remind that PRIDA aims to compute privacy-preserving data aggregation in the semi-honest adversarial model as mentioned in Section 8.4.1.

Both multi-party fully homomorphic encryption and secure two-party computation are proven to be secure. Indeed, all the three multi-party fully homomorphic encryption (MP-FHE) schemes are proved semantically secure [87, 88, 90, 91]. For the security of the new Post-process algorithm, it is crucial to add some noise, which has to be done with each partial key addition and/or removal. Note that in Algorithm 7, a fresh noise is added as a part of the MK-TFHE . Encrypt algorithm, and it is sufficient to do that once for both key addition and removal.

Input and output privacy against External Adversaries. Before sending any data, each Data Owner (DO) secret shares its choice vector **cv** and data vector **dv** and further encrypts the shares of **dv** using one of the MP-FHE schemes following the corresponding protocol version. Thanks to the semantic security guarantees of the MP-FHE schemes and the information-theoretical security of the arithmetic secret sharing, an external adversary who does not participate in PRIDA cannot obtain any information regarding inputs of Data Owners (DOs) and results of authorised Data Analysers (DAs).

Input and output privacy against Adversarial Data Owner. A semi-honest DO may wish to learn other DOs' inputs and the aggregate results of authorised DAs. Since each input is secretly shared and further encrypted with the keys of DAs, Agg1, and Agg2, and since DO does not have access to the corresponding secret keys, DO cannot discover the inputs of other DOs and the results.

Input and output privacy against Adversary Aggregator. All information that Agg1 (or Agg2) receives are encrypted under two additional keys than its key, i.e. the keys of DA and Agg2 (or Agg1). Assuming that Agg1 cannot collude with Agg2, thanks to the security guarantees of the underlying cryptosystems, an Aggregator acting curiously can neither learn inputs of DOs nor the aggregate result of authorised DAs.

Protocol 6 PRIDA based on MK-TFHE

Inputs. DO_i, $i \in \{1, \ldots, n\}$, inputs a choice vector $\mathbf{cv}_i$ and a data vector $\mathbf{dv}_i$ of size m. Also, a pre-defined threshold t and the public parameters pp are published.
Output. If $\mathrm{cv}_{total_j} \geq t$, DA_j obtains the aggregate result s_j, $j \in \{1, \ldots, m\}$. Otherwise, DA_j obtains nothing.
Protocol steps:

1. *Setup executed by DO_i, DA_j, Agg1, and Agg2.*
   (a) Generate $\mathsf{sk}_p \leftarrow \text{MK-TFHE.KeyGen}(\mathsf{pp})$ where $p = \mathrm{DO}_{i_1}, \mathrm{DO}_{i_2}$.
   (b) Generate $\mathsf{sk}_p \leftarrow \text{MK-TFHE.KeyGen}(\mathsf{pp})$ where $p = \mathrm{DA}_j$, Agg1, Agg2.
2. *Data protection executed by DO_i.*
   (a) Generate $(\langle \mathbf{cv}_i \rangle_1, \langle \mathbf{cv}_i \rangle_2) \leftarrow \text{AS.Share}(2, \mathbf{cv}_i)$.
   (b) Generate $(\langle \mathbf{dv}_i \rangle_1, \langle \mathbf{dv}_i \rangle_2) \leftarrow \text{AS.Share}(2, \mathbf{dv}_i)$.
   (c) Generate $\left(\langle \mathrm{dv}_{ij} \rangle_{k,\mathrm{DA}_j}, \langle \mathrm{dv}_{ij} \rangle_{k,\mathrm{DO}_{i_1}}, \langle \mathrm{dv}_{ij} \rangle_{k,\mathrm{DO}_{i_2}}\right) \leftarrow \text{AS.Share}(3, \langle \mathrm{dv}_{ij} \rangle_k)$ for $k = 1, 2$.
   (d) $[\langle \mathrm{dv}_{ij} \rangle_{k,p}] \leftarrow \text{MK-TFHE.Encrypt}(\langle \mathrm{dv}_{ij} \rangle_{k,p}, \mathsf{sk}_p)$ for $k = 1, 2$ and $p = \mathrm{DA}_j, \mathrm{DO}_{i_1}, \mathrm{DO}_{i_2}$.
   (e) Call MK-TFHE.Pre-process with all $[\langle \mathrm{dv}_{ij} \rangle_{k,p}]$ for $T^* = \{id_{DA_j}, id_{DO_{i_1}}, id_{DO_{i_2}}\}$.
   (f) $[\langle \mathrm{dv}_{ij} \rangle_k] \leftarrow \text{MK-TFHE.Eval}(+, ([\langle \mathrm{dv}_{ij} \rangle_{k,\mathrm{DA}_j}], [\langle \mathrm{dv}_{ij} \rangle_{k,\mathrm{DO}_{i_1}}], [\langle \mathrm{dv}_{ij} \rangle_{k,\mathrm{DO}_{i_2}}]))$ for $k = 1, 2$.
   (g) Generate random vectors $\boldsymbol{\alpha}_i$, $\boldsymbol{\beta}_i$ and compute $\boldsymbol{\gamma}_i$ such that $\boldsymbol{\gamma}_i = \boldsymbol{\alpha}_i \star \boldsymbol{\beta}_i$.
   (h) *Beaver's triplets.* Call AS.Share(., 2) for $\boldsymbol{\alpha}_i$, $\boldsymbol{\beta}_i$, $\boldsymbol{\gamma}_i$.
   (i) Send $[\langle \mathbf{dv}_i \rangle_k]$, $\langle \mathbf{cv}_i \rangle_k$, $\langle \boldsymbol{\alpha}_i \rangle_k$, $\langle \boldsymbol{\beta}_i \rangle_k$, and $\langle \boldsymbol{\gamma}_i \rangle_k$ to Agg$k$, $k = 1, 2$.
3. *Preliminary counting executed by Agg1 and Agg2.*
   (a) Locally add $\langle \mathbf{cv}_i \rangle_k$ to obtain $\langle \mathbf{cv}_{total} \rangle_k$.
   (b) Agg1 and Agg2 exchange $\langle \mathbf{cv}_{total} \rangle_k$ to get $\mathbf{cv}_{total} = (\mathrm{cv}_{total_1}, \ldots, \mathrm{cv}_{total_j}, \ldots, \mathrm{cv}_{total_m})$.
   (c) DA_j is labelled as authorised if $\mathrm{cv}_{total_j} \geq t$.
4. *Aggregation executed by Agg1 and Agg2.*
   (a) Call MK-TFHE.Post-process with known secret keys of DO_{i_k} and Agg_k to change the part of $[\langle \mathbf{dv}_i \rangle_k]$ related to DO_{i_k}'s secret key to Agg_k's secret key.
   (b) Agg1 and Agg2 exchange the resulting samples and repeat Step (a) ending up with samples encrypted with the keys of DA_j, Agg1, Agg2.
   (c) Jointly calculate the result $[s_j] = \sum [\mathrm{dv}_{ij} \cdot \mathrm{cv}_{ij}]$ for authorised DA_j.
5. *Decryption executed by Agg1, Agg2, and DA_j.*
   (a) Agg2: Send the aggregate result vector $[\mathbf{s}] = (\ldots, [s_j], \ldots)$ to Agg1.
   (b) Agg1,Agg2: Employ $\mu_k \leftarrow \text{MK-TFHE.PartialDecrypt}$ where $k = 1, 2$.
   (c) Agg1: Send μ_1 to Agg2.
   (d) Agg2: Send μ_1 and μ_2 to the authorised DA_j.
   (e) DA_j: Call $\mu_3 \leftarrow \text{MK-TFHE.PartialDecrypt}$.
   (f) DA_j: Run MK-TFHE.Merge with μ_1, μ_2, μ_3 to find s_j.

Protocol 7 PRIDA based on Th-FHE

Inputs. DO_i, $i \in \{1, \ldots, n\}$, inputs a choice vector $\mathbf{cv}_i$ and a data vector $\mathbf{dv}_i$ of size m. Also, a pre-defined threshold t and the public parameters pp are published.
Output. If $\mathrm{cv}_{total_j} \geq t$, DA_j obtains the aggregate result s_j, $j \in \{1, \ldots, m\}$. Otherwise, DA_j obtains nothing.
Protocol steps:

1. *Setup executed by DA_j, Agg1, and Agg2.*
   (a) $(\mathsf{pk}, \mathsf{sk}_{DA_j}, \mathsf{sk}_{Agg1}, \mathsf{sk}_{Agg2}) \leftarrow$ Th-FHE . KeyGen(pp).
2. *Data protection executed by DO_i.*
   (a) Generate $(\langle \mathbf{cv}_i \rangle_1, \langle \mathbf{cv}_i \rangle_2) \leftarrow \mathrm{AS}.\mathrm{Share}(2, \mathbf{cv}_i)$.
   (b) Generate $(\langle \mathbf{dv}_i \rangle_1, \langle \mathbf{dv}_i \rangle_2) \leftarrow \mathrm{AS}.\mathrm{Share}(2, \mathbf{dv}_i)$.
   (c) $[\langle \mathrm{dv}_{ij} \rangle_k] \leftarrow$ Th-FHE . Encrypt$(\langle \mathrm{dv}_{ij} \rangle_k, \mathsf{pk})$, $k = 1, 2$.
   (d) Generate random vectors $\boldsymbol{\alpha}_i$, $\boldsymbol{\beta}_i$ and compute $\boldsymbol{\gamma}_i$ such that $\boldsymbol{\gamma}_i = \boldsymbol{\alpha}_i \star \boldsymbol{\beta}_i$.
   (e) *Beaver's triplets.* Call $\mathrm{AS}.\mathrm{Share}(2, .)$ for $\boldsymbol{\alpha}_i$, $\boldsymbol{\beta}_i$, $\boldsymbol{\gamma}_i$.
   (f) Send $[\langle \mathbf{dv}_i \rangle_k]$, $\langle \mathbf{cv}_i \rangle_k$, $\langle \boldsymbol{\alpha}_i \rangle_k$, $\langle \boldsymbol{\beta}_i \rangle_k$, and $\langle \boldsymbol{\gamma}_i \rangle_k$ to Agg$k$, $k = 1, 2$.
3. *Preliminary counting executed by Agg1 and Agg2.*
   (a) Obtain $\langle \mathbf{cv}_{total} \rangle_k$ such that $\sum \langle \mathbf{cv}_i \rangle_k$.
   (b) Agg1 and Agg2 exchange $\langle \mathbf{cv}_{total} \rangle_k$ to get $\mathbf{cv}_{total} = (\mathrm{cv}_{total_1}, \ldots, \mathrm{cv}_{total_j}, \ldots, \mathrm{cv}_{total_m})$.
   (c) DA_j is labelled as authorised if $\mathrm{cv}_{total_j} \geq t$.
4. *Aggregation executed by Agg1 and Agg2.*
   (a) Jointly compute $[\mathbf{s}] = \sum [\mathbf{dv}_i \star \mathbf{cv}_i]$ for each authorised DA_j.
5. *Decryption executed by Agg1, Agg2, and DA_j.*
   (a) Agg2: Send the aggregate result vector $[\mathbf{s}] = (\ldots, [s_j], \ldots)$ to Agg1.
   (b) Agg1, Agg2: Employ $\mu_k \leftarrow$ Th-FHE . PartialDecrypt to decrypt $[\mathbf{s}]$ where $k = 1, 2$.
   (c) Agg1: Send μ_1 to Agg2.
   (d) Agg2: Send μ_1 and μ_2 to the authorised DA_j.
   (e) DA_j: Call $\mu_3 \leftarrow$ Th-FHE . PartialDecrypt.
   (f) DA_j: Run Th-FHE . Merge with μ_1, μ_2, μ_3 to find s_j.

Input and output privacy against Adversary Data Analyser. We now consider a semi-honest DA who is interested in discovering either some DOs' individual inputs or some other DAs' received outputs. We remind that all exchanged data in PRIDA are secretly shared or shared-encrypted or multi-party encrypted. An adversarial DA cannot discover the inputs of DOs thanks to the use of the semantically secure MP-FHE and the information-theoretically secure Arithmetic secret sharing. The only information that DA can decrypt is the outcome of the aggregation if authorised by at least t DOs. On the other hand, if outputs and inputs are destined to other DAs and hence encrypted with their keying material in addition to the two Aggregators' keys, the adversarial DA becomes an external adversary as it does not play any role in this aggregation protocol.

Control of Data Owners on Data Analysers. With a choice vector, **cv**, each DO can easily decide which DA can receive its data while computing the data aggregation. DO secret shares its choice vector **cv** into two and sends one share to one Aggregator. Thanks to *the preliminary counting phase*, the non-authorised DA cannot receive the aggregate result. Moreover, even if a non-authorised DA colludes with Agg2, they cannot retrieve the aggregate result because Agg1 has already terminated the data aggregation for this non-authorised DA. Further, the actual data aggregation result remains bogus since bogus data is assigned to this specific element when $\mathrm{cv}_{ij} = 0$. Moreover, thanks to MP-FHE, Agg2 and DA need Agg1 to partially decrypt the encrypted aggregate result during *the decryption phase*.

Anonymity of Data Owners. A Data Analyser DA_j can discover the aggregate result without learning the individual DO if and only if the number of DOs should be at least a pre-defined threshold t during *the preliminary counting phase*. Otherwise, DA learns nothing.

Input and output privacy against collusions between Data Owner and Aggregator. The collusions of Agg1 with one DO only results in discovering the individual input of the actual DO and the other secret shared values. Hence all remaining information are either secret shared, or shared-encrypted or multi-key encrypted with two keys for which they do not have the corresponding secret key.

Collusions between Data Analyser and Aggregator. When an authorised DA and Agg2 collude, they do not discover any leakage regarding inputs of DOs and/or outputs of other DAs. Indeed, even if the Agg2 aggregates the result for an unauthorised DA and forwards it, the only information that DA would be able to decrypt would be bogus (DOs encrypt bogus data for unauthorised DAs).

8.4.4 Performance Evaluation

We propose to evaluate the performance of PRIDA using a use case scenario whereby multiple Data Analysers (DAs) as pharmaceutical companies wish to discover the side effects of some medicine on multiple patients who take the underlying medicine, using some statistics such as sum, average, etc. over this target group. In this scenario, when patients acting as Data Owners (DOs) are willing to participate in this discovery with their

private data, patients use multi-party FHE and 2PC to protect their data before sending them to the non-colluding two Aggregators. Once the Aggregators receive these protected data, they jointly perform the required operations over these data and send the encrypted aggregate result to the authorised pharmaceutical companies if chosen by enough patients. Later on, these companies decrypt the result. Therefore, we conduct several experiments with different numbers of DOs and DAs and analyse the computational cost at each party.

Experimental Setup

We have implemented each version of PRIDA. For Protocol 5. For PRIDA v1, we have implemented the asymmetric MK-FHE solution described in [87] using the BFV [60, 66] and CKKS [62] schemes from the SEAL library v3.6.4 [108].

PRIDA v2, which is based on MK-TFHE, is implemented using an extended version of the MK-TFHE library [89]: We have implemented all binary gates except for the NAND gate, which was already available; moreover, we have revisited the original MK-TFHE . Encrypt and MK-TFHE . Decrypt algorithms, because they require all the keys from the very beginning. Thus, in order to prevent a significant increase in the number of keys and consequently the size of the ciphertext. Accordingly, we have split the two algorithms into MK-TFHE . Encrypt, MK-TFHE . Pre-process, MK-TFHE . PartialDecrypt and MK-TFHE . Merge, and implemented the new MK-TFHE . Post-process algorithm.

In addition, we have implemented some optimisations: (i) We have implemented the double-and-add algorithm for the multiplication of an MK-TFHE-encrypted number by a known scalar; and (ii) we have implemented a variant of the addition algorithm, which employs ternary logical gates and it only requires 2 bootstrappings instead of 5, hence saving about a factor of 2.5 time.

Finally, PRIDA v3 based on Th-FHE is implemented with the PALISADE library v1.10.6 [84]. Once again, similar to the implementation of Protocol 5, both BFV and CKKS are used for Protocol 7.

We have followed the standard HE security recommendations (e.g., 128-bit security) indicated in [221] for all three protocols. We implemented the 2PC protocol on our own: Data is shared by generating a random number first and then computing the other share accordingly: Additions of shares are performed locally, and multiplications are implemented according to the Beaver triplets' algorithm [39]. All experiments have been carried out using a desktop computer with 3.5 GHz Intel Core i7-7800X processor, 128 GB RAM, and the Ubuntu 20.04.2 LTS operating system. All parties are emulated in the same environment.

Performance results

We have first evaluated the computation cost of each PRIDA player in a scenario with 100 DOs and 2 DAs. We realised that the computation time resulting from PRIDA v2 was very significant and hence run Protocol 2 in a simpler scenario with 3 DOs and 2 DAs. The results are shown in Table 8.1. These values correspond to an average of measurements from ten executions. We first observe that for the three versions of PRIDA, only Aggregators perform costly operations. Indeed, while Aggregator2 takes 25.73

Table 8.1 – Performance results for each player of PRIDA (computation time in s).

Protocols	*Data Owner*	*Data Analyser*	*Aggregator1*	*Aggregator2*
Protocol 5 with MK-BFV	0.299	0.091	19.878	26.782
Protocol 5 with MK-CKKS	0.294	0.071	20.234	25.730
Protocol 6 with MK-TFHE	4.452	2.275	2632.251	3160.709
Protocol 7 with Th-BFV	0.009	0.224	1.268	1.294
Protocol 7 with Th-CKKS	0.009	0.345	1.050	1.096

(26.78) seconds within PRIDA v1 with MK-CKKS (with MK-BFV), 3160.71 seconds with PRIDA v2 (i.e. with MK-TFHE), and 1.10 (1.29) seconds within PRIDA v3 with Th-CKKS (or with Th-BFV), Aggregator1 takes 20.23 (19.88) seconds, 2632.25 seconds, and 1.05 (1.10) seconds, respectively. We observe that there is a slight difference on the computation time between Aggregator1 and Aggregator2. We believe that this difference originates from the different workload attributed to each Aggregator: Aggregator2 indeed performs additional operations to finalise the aggregation phase (see Algorithm 6).

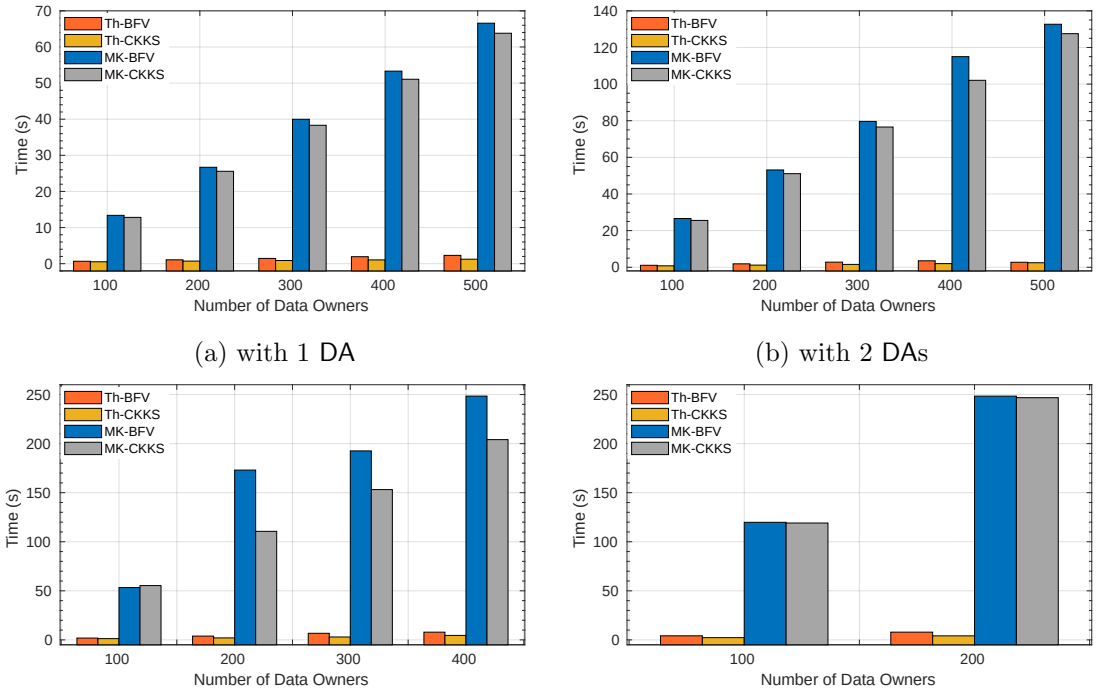
Finally, the computation time of the DA corresponds to the time that DA takes to execute the setup phase and the decryption of the aggregate result.

We also observe that PRIDA v2 takes more time than the two other protocols. We believe that the overhead is caused mainly due to the additional layers of encryption, for which the MK-TFHE library is not optimised yet. Also note that the MK-TFHE Library is—unlike the SEAL or PALISADE Library—only an experimental code.

In order to study the scalability of PRIDA, we have also measured the time of the actual aggregation operation in PRIDA (namely, the preliminary counting phase, the aggregation phase, and the decryption phase) with respect to a larger number of DOs and an increasing number of DA s. Since The PRIDA v2 cost is high, we propose to study PRIDA v1 and PRIDA v3 only. Detailed results are shown in Fig. 8.3[2]. We remind that PRIDA enables DOs to select/control which DA can receive the aggregate result for which they are contributing. For each experiment, the anonymity threshold t is set to be 50. In order to consider all possible scenarios with respect to the authorisation aspects, we propose that in Fig. 8.3b, DOs authorise each DA uniformly randomly whereas, in Fig. 8.3c, each DO authorises three out of five DA s (corresponding to a ratio of 50%). Lastly, in Fig. 8.3d, DOs authorise nine DA s out of ten.

We observe that this computation time linearly increases with the number of DOs and the number of DA s. We also observe that there is a significant difference between

[2]These results correspond to the average from three executions.



(a) with 1 DA

(b) with 2 DAs

(c) with 5 DAs

(d) with 10 DAs

Figure 8.3 – Data aggregation time for Protocol 5 (MK-FHE) and Protocol 7 (Th-FHE) using BFV and CKKS

the execution times of PRIDA v1 and PRIDA v3. We believe that one of the reasons for this difference is due to the fact that PALISADE is implemented in a multi-threading way, whereas SEAL is single-threaded. Furthermore, since PRIDA v1 relies on the use of multiple keys, the execution time of the underlying MK-FHE . Eval algorithm linearly increases with the number of keys. On the contrary, in PRIDA v3, the cost of Th-FHE . Eval can be considered as approximately equal to Eval with one key only (i.e., the encryption algorithm uses one single public key). On the other hand, PRIDA v1 does not require a complicated setup phase since, as opposed to PRIDA v3, each party independently generates its own keying material.

To study the scalability feature of PRIDA even more, we have run PRIDA v1 and PRIDA v3 with 1000 Data Owners and 1 or 2 Data Analysers. As depicted in Fig. 8.4, we observe that privacy-preserving data aggregation operations remain possible. Further, PRIDA is more realistic than the existing solutions and can be extended for the applications of federated learning, which requires data aggregation.

To summarise, while the fastest PRIDA version is Th-FHE with 2PC based solution; yet, it requires the two Aggregators and each Data Analyser to jointly generate a common public key. On the other hand, PRIDA v1 and PRIDA v2 do not need a jointly generated unique key, but on the other hand, DOs need to do more operations to protect their input privacy. The cost of PRIDA v2 remains significant. We believe that this is mainly due to

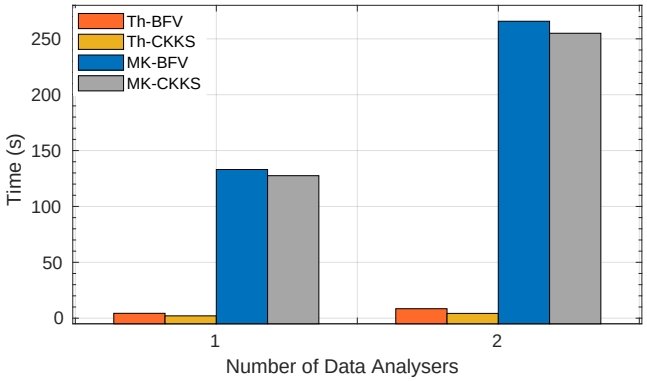


Figure 8.4 – PRIDA scalability

the experimental prototype of the MK-TFHE library and may be improved in the future.

8.5 Conclusion of privacy-preserving data aggregation

In this chapter, we have introduced our new privacy-preserving data aggregation solution, namely PRIDA, that enable two non-colluding Aggregators (i.e., simply cloud servers) to efficiently aggregate input data coming from many Data Owners and obtain the aggregate information that is/are interesting for multiple Data Analysers.

PRIDA is implemented on top of the privacy-by-design approach for privacy-preserving data aggregation that combines multi-party fully homomorphic encryption (MP-FHE) with secure two-party computation. Thanks to the use of these two cryptographic building blocks with a setting involving two non-colluding Aggregators, PRIDA supports scenarios with more than one Data Analyser. MP-FHE is instantiated with three different schemes, multi-key fully homomorphic encryption (PRIDA v1), multi-key TFHE (PRIDA v2), and threshold fully homomorphic encryption (PRIDA v3). Furthermore, PRIDA enables Data Owners to have some control over which Data Analyser can have access to the resulting aggregated information. Moreover, with the introduction of an anonymous counting phase, Data Analysers can discover the aggregation result only when a sufficient number of Data Owners (i.e., more than a pre-defined threshold) authorise them. We have provided a detailed security analysis of PRIDA considering different potential adversaries, including potential collusions among parties. Our experimental results seem promising in terms of scalability.

We introduce the notion of multi-party homomorphic encryption and further instantiate it with three different advanced homomorphic encryption schemes: MK-FHE, which allows the execution of operation over multiple data encrypted with different keys; MK-TFHE, which is the symmetric version of multi-key homomorphic encryption and therefore requires a dedicated key management setup phase; and Th-FHE, which requires the collaboration of multiple parties to generate one unique public key. Moreover, we propose a new algorithm, MK-TFHE . Post-process, to lower the execution of

MK-TFHE . Pre-process since each Data owners have two keys and the Aggregators need to make all the encrypted data encrypted with all keys.

Chapter 9

Conclusion Remarks and Future Research

One of the most underrated secrets to success is to start before you're ready.

Marie Forleo

9.1 Summary

In this thesis, we have studied the design of privacy-preserving variants of machine learning techniques. We observe that such a design work is not straightforward even when using advanced cryptographic techniques such as homomorphic encryption and secure multiparty computation, and these learning techniques should be customised to be compatible with the underlying cryptographic techniques. The modifications performed on the original learning techniques usually have an impact on the efficiency and utility/accuracy of these techniques. Hence, in this thesis, we have proposed to design and develop new privacy-preserving machine learning techniques that keep data confidential, and at the same time, features a good level of accuracy and performance.

We have investigated three machine learning techniques: (i) Neural networks, which help build models to make accurate prediction or classification; (ii) (Trajectory) Clustering techniques that allow the regrouping of similar (trajectory data) data items; and (iii) Data Aggregation that enables any party to collect data from multiple sources and perform some (statistical) analytics such as sum or average over the collected data. We investigated the suitability of existing cryptographic techniques such as homomorphic encryption or secure multiparty computation to the actual three ML techniques. We

further came up with the design of their privacy-preserving variants. We studied data privacy in two different scenarios:

The first one is a single-server scenario whereby a cloud server, or particularly a model provider, has or receives a neural network model [93–96], and a querier wishes to use the model to classify its input data. In the single-server scenario, we have designed several solutions based on two cases: (i) The employed neural network model is either in the clear form ([93, 94, 96] based on 2PC or PHE and 2PC, see Chapter 4) or in the encrypted form ([95] based on H-PRE, see Chapter 4) to fully utilise the advantages of the cloud server, and therefore, the workload of the model provider is minimised; and (ii) a better look into the cryptographic techniques' suitability when their integration with neural networks without any impact on data privacy, the performance of the designs, and accuracy of the proposed private solutions. In the single-server setting, we also proposed a privacy-preserving trajectory clustering solution for TRACLUS [162] utilising 2PC: We assume that a data owner has multiple trajectories and would like to cluster them into similar line segments. Therefore, the data owner privately shares the underlying data and outsources one share of them to a cloud server. They both run the private protocol interactively to obtain the partial result. The cloud server sends its partial result to the data owner, who can bring those results and find the clusters. Note that in this solution, the data owner needs the same equipment, particularly expertise in machine learning and cryptographic techniques and computational resources.

We study a second scenario that involves more than one cloud server, namely the two-server scenario, to decrease the load of the clients' or data owners' duties expected from the first scenario. We assume that a data owner (or a client) has some data and would like to join some learning techniques (neural network classification, trajectory clustering, or data aggregation), but this data owner has neither clustering-specific expertise nor computational resources. Once again, we have designed privacy-preserving solutions for neural network classification, TRACLUS, and data aggregation to meet the need of the querier/data owner: The data owner privately shares the data and outsources them to two non-colluding cloud servers. These two cloud servers run the private protocol interactively to obtain the partial result. These partial results are sent to the data owner, who can bring those results to obtain the learning result. In these solutions [94, 162, 220], we achieve the computational costs of the clients/data owners and envoy these costs to the two cloud servers. We have also investigated enabling fully private machine learning techniques and reducing the computations at the cloud servers' side with a balance between privacy, efficiency, and the result quality evaluation.

In order to put our solutions for neural network classification, TRACLUS, and data aggregation into some colour and compare them, in Figures 9.1 and 9.2, we try to allocate our solutions with respect to their privacy and accuracy/quality evaluation, and their privacy and efficiency levels.

When considering the accuracy/quality evaluation of the proposed solutions, since we use some approximations for complex operations and encodings, which enable the underlying machine learning techniques to be compatible with cryptographic techniques, PAC, SwaNN, and ProteiNN result in similar accuracy. Yet, for the privacy level, the neural network model in PAC and SwaNN is in the cleartext form on the cloud

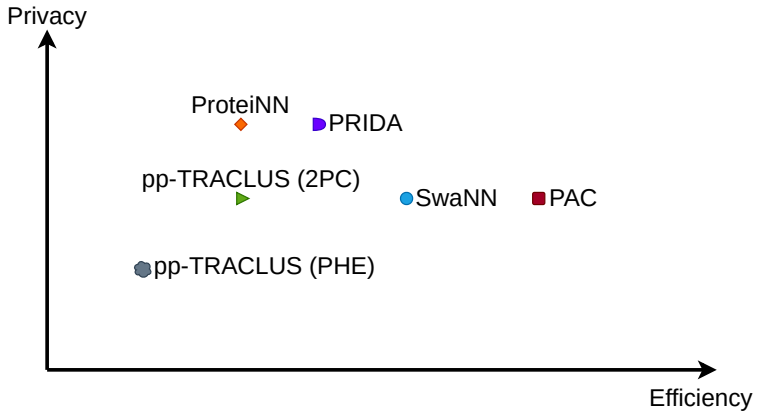


Figure 9.1 – All the proposed privacy-preserving machine learning techniques in this thesis regarding the trade-off between privacy and efficiency

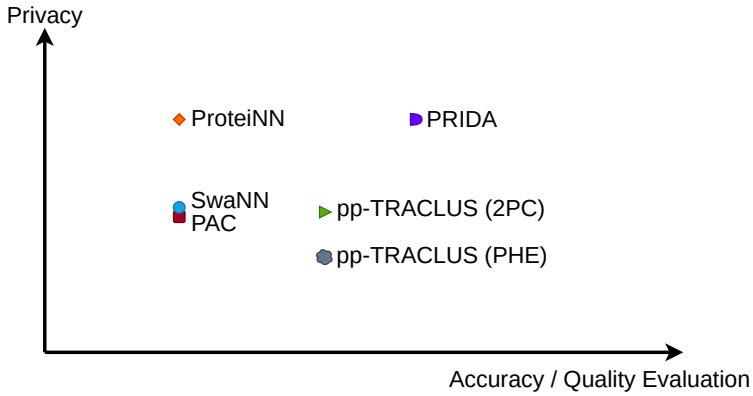


Figure 9.2 – All the proposed privacy-preserving machine learning techniques in this thesis regarding the trade-off between privacy and accuracy/quality evaluation

server whereas the model in ProteiNN is encrypted on the cloud server. The solutions for TRACLUS use only approximations for the TRACLUS complex operations, and therefore, the quality evaluation is better than PAC, SwaNN, and ProteiNN; however, since the PHE-based pp-TRACLUS leaks some information regarding the clusters, the privacy level is lower than the other solutions. Because PRIDA does not employ any approximations or encodings, it is better in data privacy and accuracy/quality evaluations than others. For the performance results, the neural network classification solutions seem better than others while the PHE-based TRACLUS solution is the slowest one (due to the several iterations when considering line segments being neighbours).

9.2 Future Work

We employed several cryptographic techniques to develop privacy-preserving neural network classification, privacy-preserving trajectory clustering, and privacy-preserving data aggregation protocols. Possible future research directions to investigate can be listed as below:

- The research studies in this thesis have been presented in the semi-honest security model, and they can be extended to address more powerful adversaries such as malicious adversaries. In order to design secure protocols against the latter adversaries, zero-knowledge proofs can be utilised. By executing zero-knowledge proofs for every message, parties involved in the protocol neither learn any more information than the semi-honest security model nor malicious parties can cheat thanks to soundness.
- Each designed and developed privacy-preserving protocol can be analysed with formal methods. For example, the symbolic analysis using computer-aided cryptography (such as employing analysis tool Tamarin [222]) can be studied: Potential adversaries and/or possible collusions between involved parties can be easily defined. Thus, more secure protocols can be designed according to the analysis tool(s).
- Lastly, defense mechanisms against attacks targeting model privacy (such as model inversion, membership inference attacks, etc.) can be investigated. Such attacks can help the adversary discover the underlying training data in the context of neural networks, and hence, once again harm the privacy of some individuals that are involved in this dataset.

Appendices

Appendix A

Résumé Français

What is done in love is done well.

Vincent van Gogh

L'utilisation des appareils de l'Internet des objets (IoT) partout : chez nous, sur nos poignets ou dans nos poches, et le développement d'applications conviviales pour ces appareils IoT ont encouragé les gens à une utilisation généralisée. De plus, avec l'évolution des technologies de cloud computing, les entreprises axées sur les données, par exemple les fournisseurs d'applications ou de services, *collectent et stockent* facilement une quantité massive de données. Une telle *abondance de données* permet de dériver des informations pertinentes sur leurs utilisateurs grâce à des analyses avancées telles que *analyse statistique* (somme, moyenne, etc.) ou *techniques d'apprentissage automatique* (réseaux de neurones, clustering, etc.). Les résultats analytiques peuvent aider les entreprises à améliorer leurs services clients existants ou à en proposer de nouveaux.

Les entreprises sont également attirées par le partage de ces données avec leurs partenaires tiers. Cependant, le paradigme *données collectées/traitées/partagées* soulève de *sérieux problèmes de confidentialité* principalement en raison de *la grande sensibilité des données*. Lorsque les entreprises essaient d'en tirer de la valeur, elles sont confrontées à des défis croissants avec *garantir les garanties de confidentialité des données* et le respect des réglementations sur la protection des données [2, 3]. De plus, la collecte, le traitement et le partage des données sous-jacentes peuvent entraîner une violation de la vie privée des individus (par exemple, la divulgation de la date, du lieu ou de la participation à un événement social). Il existe des exemples récents d'atteintes à la vie privée provenant de ce type d'utilisation d'applications: selon deux nouvelles du Guardian, (i) certains documents fournis par Edward Snowden en 2014 ont révélé que la NSA avait utilisé le jeu mobile Angry Birds pour collecter des données d'utilisateurs telles que comme l'âge,

le sexe et l'emplacement[1]; et (ii) le scandale des données Facebook-Cambridge Analytica au début de 2018 a fait la une des journaux lorsqu'il a été révélé que la société de conseil avait collecté les données de Facebook pour profiler les électeurs américains[2].

A.1 Apprentissage automatique en tant que service

La technologie du cloud computing et le succès des techniques d'apprentissage automatique conduisent à un changement de paradigme dans les services technologiques qui permettent aux entreprises axées sur les données de déléguer leurs tâches d'apprentissage automatique à des serveurs cloud dotés d'une expertise spécifique au domaine en apprentissage automatique et en ressources de calcul pour les analyses requises. *Apprentissage automatique en tant que service* (MLaaS) est l'un de ces services qui permet aux entreprises d'effectuer des tâches d'apprentissage automatique sur le serveur cloud. Le MLaaS permet à ces entreprises d'externaliser leurs tâches d'apprentissage automatique sur une plate-forme cloud [1].

A.2 Confidentialité des données vs techniques d'apprentissage automatique

La vie privée d'un individu est l'un des droits fondamentaux. Le MLaaS sur les données provenant d'individus est devenu de plus en plus attrayant pour les entreprises axées sur les données avec la capacité croissante de traitement des données (collecter, effectuer des analyses de données avancées sur les données, etc.). Pourtant, les données collectées, stockées, traitées ou partagées sont généralement des données sensibles à la confidentialité et, ces dernières années, de nombreuses réglementations en matière de protection des données telles que le règlement général européen sur la protection des données (RGPD) ou ePrivacy [2, 3] ont été a émergé pour protéger la vie privée des individus. Ces règles de protection des données garantissent que les données traitées sont protégées et ne divulguent la vie privée d'aucune personne. L'aspect *privacy-by-design* du RGPD peut être appliqué en utilisant des techniques cryptographiques pour prendre en charge la protection des données et, en même temps, l'utilisation de techniques d'apprentissage automatique sur ces données protégées.

Les techniques d'apprentissage automatique peuvent être considérées comme un système conçu pour apprendre ou résoudre des problèmes en fonction des observations de son environnement. Les techniques d'apprentissage automatique sont des algorithmes mathématiques capables de résoudre des problèmes complexes liés à l'informatique. Aujourd'hui, plusieurs techniques d'apprentissage automatique telles que le réseau de neurones ou le clustering sont bien connues de tous, qu'il soit data scientist ou non. Dans cette thèse, nous nous concentrons sur trois techniques:

[1] https://www.theguardian.com/world/2014/jan/27/nsa-gchq-smartphone-app-angry-birds-personal-data

[2] https://www.theguardian.com/news/2018 /mar/17/cambridge-analytica-facebook-influence-us-election

1. *Réseau neuronal* est inspiré du cerveau humain composé de nombreux neurones connectés qui servent certaines fonctionnalités au corps humain [5, 6]. Un réseau de neurones se compose de deux phases : une phase d'apprentissage (ou phase d'apprentissage), où le modèle de réseau de neurones est construit en apprenant/en acquérant de nouvelles capacités à partir des données de l'ensemble de données d'apprentissage ; et une phase de classification (ou appelée phase de prédiction ou d'interrogation), dans laquelle le réseau neuronal créé est testé avec de nouvelles données de l'ensemble de données de test (c'est-à-dire différentes des éléments de données dans l'ensemble de données d'apprentissage). Le modèle de réseau de neurones peut être défini comme la composition de plusieurs fonctions prenant en entrée des matrices et/ou des vecteurs qui sont construits pendant la phase d'apprentissage : Ces matrices et vecteurs sont créés en déterminant l'influence d'une entrée de neurone donnée sur la sortie de ce neurone. La phase de classification utilise le modèle construit en prenant des données de test et en produisant une étiquette dessus. Dans cette thèse, nous utilisons le jeu de données d'arythmie MIT-BIH[3] pour classer les données de rythme cardiaque. De plus, nous nous concentrons également sur les réseaux de neurones convolutifs (CNN) qui sont utilisés pour classer les images à l'aide des ensembles de données MNIST et CIFAR-10 [223, 224]. Un réseau de neurones se compose généralement de trois couches différentes : la couche d'entrée, la ou les couches cachées et la couche de sortie. Pour les couches cachées, les plus utilisées sont la couche convolutive (si une classification d'image est nécessaire), la couche entièrement connectée, la couche d'activation et la couche de mise en commun. Ces couches peuvent être considérées comme des fonctions spéciales telles que la multiplication matrice-vecteur, le calcul de `Max`, ou le calcul de `Sigmoïde`, vers un réseau de neurones. Notez que l'ensemble de données de test permet également de déterminer la précision du réseau de neurones en utilisant des éléments de données dans l'ensemble de données de test alimenté au réseau de neurones construit dans la phase d'apprentissage pour mesurer sa précision, c'est-à-dire comment classer correctement les données sous-jacentes. Pour plus de détails, nous renvoyons les lecteurs aux Chapitres 2, 4 et 6.

2. *Regroupement de trajectoires* (TRACLUS) est un algorithme de clustering basé sur la densité [7] conçu et optimisé pour le clustering de trajectoires. TRACLUS se compose de deux phases : Une phase de partitionnement, dans laquelle les trajectoires sont divisées en sous-trajectoires, à savoir des segments de droite (représentés par deux points) aussi proches que possible des trajectoires d'origine ; et une phase de regroupement (ou appelée phase de regroupement), où les sous-trajectoires segmentées sont regroupées en certains groupes en fonction de leurs similitudes, c'est-à-dire qu'elles sont voisines de certains segments de ligne. Ces phases de TRACLUS contiennent plusieurs fonctions spéciales telles que le calcul du logarithme, la fonction sinus, la division, etc. Pour TRACLUS, nous utilisons plusieurs ensembles de données tels que les ensembles de données Hurricane, Deer, Taxi ou Travel [7] pour regrouper des éléments de données dans le même cluster

[3]https://www.physionet.org/physiobank/database/mitdb/

ou étiquetez-les comme bruit non-cluster. Pour évaluer la qualité de regroupement de TRACLUS, nous utilisons plusieurs mesures d'évaluation de la qualité pour déterminer que nos résultats sont significatifs (voir les Chapitres 2, 5 et 7).

3. *Agrégation de données* est le processus consistant à assembler des données, à les présenter sous une forme résumée et à effectuer des analyses statistiques telles que la somme ou la moyenne sur ces données. Les données sous-jacentes proviennent généralement de plusieurs sources de données (pouvant être des individus ou des pools de données de plusieurs entreprises axées sur les données) pour les réunir afin d'essayer des résultats significatifs en utilisant des analyses de données statistiques. L'agrégation de données est l'un des processus de données les plus utilisés dans le domaine de la finance (commerce de détail, investissement, etc.), de l'industrie du voyage, des réseaux de capteurs ou des moteurs de recherche [8]. Cette thèse utilise l'agrégation de données pour obtenir l'opération de somme sur les données provenant de plusieurs sources de données, et plusieurs analyseurs de données souhaitent cette sommation (voir les Chapitres 2 et 8).

Les réglementations sur la protection des données telles que le RGPD imposent que les données traitées soient confidentielles et privées lorsque les entreprises utilisent des techniques d'apprentissage automatique (ou en tant que tâche de MLaaS sur une plate-forme cloud) ; cependant, ces techniques ne peuvent pas fonctionner correctement sans avoir accès aux données sous-jacentes. Par conséquent, afin de garder les données confidentielles et, en même temps, de permettre à ces techniques de fonctionner correctement sur ces données sensibles à la confidentialité, on peut tirer parti de techniques cryptographiques telles que le cryptage homomorphe ou le calcul multipartite sécurisé.

A.3 Protocoles préservant la confidentialité pour les techniques d'apprentissage automatique

Afin de construire un protocole de préservation de la confidentialité pour les techniques d'apprentissage automatique mentionnées précédemment, il convient d'identifier les principales exigences en matière de confidentialité. Nous considérons donc deux scénarios:

1. *Techniques d'apprentissage automatique à serveur unique préservant la confidentialité* par lesquelles une partie, à savoir un demandeur (ou un propriétaire de données), a une entrée (ou a déjà collecté des données provenant de ses clients). Le demandeur/propriétaire des données souhaite déduire certaines informations pour sa saisie en utilisant les services de MLaaS fournis par une partie non fiable mais puissante, à savoir un serveur cloud. En raison du manque d'expertise spécifique au domaine en matière d'apprentissage automatique et/ou de ressources de calcul, le demandeur/propriétaire des données sous-traite ses entrées et les calculs pertinents au serveur cloud. Une fois qu'ils les reçoivent, le serveur cloud effectue les calculs requis sur les données externalisées et obtient une sortie qui sera envoyée à la partie autorisée.

2. *Techniques d'apprentissage automatique à deux serveurs préservant la confidentialité* par lesquelles un demandeur (ou un propriétaire de données) a une entrée (ou a déjà collecté des données provenant de ses clients). Le demandeur/propriétaire des données souhaite obtenir des informations en utilisant son entrée à l'aide de deux serveurs cloud mettant en œuvre les tâches MLaaS. Avec ces deux serveurs cloud, nous visons à utiliser pleinement les avantages des serveurs cloud et, par conséquent, à minimiser la charge du demandeur/propriétaire des données lors du calcul de la technique d'apprentissage automatique requise.

Dans ces scénarios, pour être conforme aux réglementations sur la protection des données et les données externalisées étant sensibles à la confidentialité, le demandeur/propriétaire des données doit être protégé. La première exigence à atteindre est de garantir la *confidentialité des entrées* contre toute partie non autorisée pendant sa durée de traitement (c'est-à-dire de sa collecte à son analyse). La partie non autorisée peut être le serveur cloud, l'autre partie dans le protocole ou une partie externe, qui ne joue aucun rôle pendant le protocole. La deuxième exigence est d'assurer la *confidentialité de sortie* contre toute partie non autorisée, car la sortie des techniques d'apprentissage automatique requises sur les données d'entrée sensibles à la confidentialité peut révéler des informations sur le demandeur/propriétaire des données. Enfin, lorsque la technique d'apprentissage automatique est le réseau de neurones, le modèle de réseau de neurones doit rester privé contre toute partie, à l'exception de son propriétaire, car le modèle sous-jacent peut divulguer des informations confidentielles sur les données d'entraînement et, par conséquent, cela peut indiquer l'identité de un individu. Notez que dans certains cas (voir Section 4.6), le modèle doit rester privé même contre le serveur cloud lui-même.

La mise en œuvre de techniques d'apprentissage automatique sur des données confidentielles nécessite l'utilisation de techniques cryptographiques telles que le cryptage (entièrement) homomorphe (F(HE)) ou le calcul sécurisé multi(deux) parties (MPC/2PC) ; cependant, ils encourent malheureusement un surcoût non négligeable en ce qui concerne les coûts de calcul et de communication. Pour combiner efficacement les techniques cryptographiques sous-jacentes aux techniques d'apprentissage automatique, la conception de ces dernières doit être revisitée : des techniques d'apprentissage automatique personnalisées doivent être conçues. Étant donné que de telles personnalisations ont un impact sur la précision (ou l'évaluation de la qualité) des techniques d'apprentissage automatique sous-jacentes, l'objectif des protocoles d'apprentissage automatique préservant la confidentialité est d'aborder le compromis entre la confidentialité, l'efficacité et l'évaluation de la précision/qualité. Par conséquent, nous pouvons détailler quatre défis lors de la construction de protocoles de préservation de la confidentialité pour le réseau de neurones, TRACLUS et l'agrégation de données:

– *Défi 1: Phases de la technique sous-jacente.* Un réseau de neurones contient deux phases, à savoir les phases d'apprentissage et de classification. Lors de l'examen de leurs fonctions ou opérations complexes dans ces phases, nous nous concentrons sur la classification des réseaux de neurones et supposons que le modèle a été construit sur des données en clair pendant la phase d'apprentissage. La phase d'apprentissage est composée de plusieurs itérations du processus des mêmes fonctions jusqu'à avoir

un bon niveau de précision. Par rapport à la phase de classification, lors de la phase de formation sur le cloud, il est parfois nécessaire de régler certains paramètres ou d'augmenter le nombre de fonctions d'activation, et par conséquent, ce besoin implique plusieurs interactions entre le propriétaire des données et le serveur cloud pour permettre au propriétaire des données pour les décider. De plus, lorsque l'on considère TRACLUS, la phase de partitionnement est plus complexe : elle nécessite une interaction entre le propriétaire des données et le client. Nous nous concentrons donc sur la phase de regroupement (ou appelée clustering) et supposons que le propriétaire des données a déjà exécuté la phase de partitionnement sur des trajectoires de texte en clair et les a divisées en segments de ligne.

– *Défi 2: Plusieurs sources de données.* Un autre défi est la multiplicité des sources de données. Les données collectées à partir de plusieurs sources posent des problèmes de confidentialité des données en raison de la multiplicité des sources de données, et chacune d'entre elles nécessite sa propre confidentialité et doit être protégée individuellement.

– *Défi 3: Opérations complexes.* Les techniques cryptographiques existantes sont incompatibles pour prendre en charge les opérations complexes d'apprentissage automatique telles que la fonction d'activation `Sigmoid` dans le réseau de neurones. Afin de garantir la confidentialité des données, les opérations complexes sont généralement approchées en certaines opérations linéaires (telles que des polynômes de faible degré) que les techniques cryptographiques peuvent prendre en charge efficacement.

– *Défi 4: Nombres réels.* Les réseaux de neurones ou TRACLUS calculent des nombres sur des nombres réels alors que les techniques cryptographiques fonctionnent sur des nombres binaires ou entiers. Par conséquent, ces nombres réels doivent être approximés/codés dans les binaires ou les entiers pour être compatibles avec la technique cryptographique sous-jacente. Cependant, des codages tels que des troncatures limitant le nombre de parties fractionnaires peuvent avoir un impact sur l'évaluation de la précision/qualité de la technique d'apprentissage automatique requise.

Pour conclure, lors de la conception de variantes préservant la confidentialité des techniques d'apprentissage automatique, il convient de prendre en compte les exigences de sécurité ou les limitations en termes de coût de calcul et de communication pour une partie interrogeant, à savoir le demandeur ou le propriétaire des données, et la partie informatique qui est le cloud serveur. De plus, les opérations complexes de la technique d'apprentissage automatique demandée doivent être approximées en certaines fonctions linéaires si elles ne peuvent pas être facilement compatibles avec les techniques cryptographiques. Au moment de décider du choix des techniques cryptographiques, les solutions basées sur MPC (ou 2PC) sont efficaces en termes de coût de calcul, par exemple pour les réseaux de neurones, tandis que les solutions basées sur (F)HE sont coûteuses en temps de calcul. Dans les solutions basées sur (F)HE, il n'y a pas d'interaction entre

le client et le serveur cloud. En revanche, dans les solutions basées sur MPC, le client et le serveur cloud calculent ensemble en toute sécurité toutes les opérations du réseau neuronal, et l'utilisation de MPC entraîne des coûts de bande passante supplémentaires. De plus, les solutions basées sur (F)HE ne prennent en charge que les opérations linéaires, mais avec MPC, les opérations linéaires et non linéaires sont calculées. Ainsi, les solutions basées sur (F)HE sont moins précises en raison de l'utilisation d'opérations linéaires uniquement ; avec MPC, la précision est bien meilleure que les solutions basées sur (F)HE. De plus, combiner (F)HE et MPC peut être une bonne solution pour obtenir de meilleurs coûts de calcul et de communication et pour obtenir une meilleure précision tout en garantissant la confidentialité des données (voir les Sections 4.5 et 6.3).

A.4 Contributions

Cette thèse étudie l'adéquation de plusieurs techniques cryptographiques aux réseaux de neurones, TRACLUS et à l'agrégation de données, à savoir : le calcul sécurisé à deux parties, le chiffrement homomorphe et le rechiffrement de proxy homomorphe, et propose plusieurs protocoles de préservation de la confidentialité pour les techniques d'apprentissage automatique sous-jacentes. en appliquant quelques approximations sur leurs opérations sous-jacentes. Nous divisons nos solutions en deux scénarios différents en fonction du nombre de serveurs cloud.

Dans la première partie de cette thèse, nous avons conçu, développé et mis en œuvre une variante préservant la confidentialité assistée par un seul serveur des techniques d'apprentissage automatique, à savoir les réseaux de neurones et le clustering de trajectoires. Nous présentons quatre solutions dans cette partie, à savoir:

1. PAC, comme le montre la Figure A.1, est une solution pour concevoir une classification des arythmies préservant la confidentialité qui préserve la confidentialité des données de battement de cœur des demandeurs par rapport au serveur cloud et du modèle de réseau neuronal aux demandeurs. Comme étude de cas, nous avons conçu un nouveau modèle basé sur l'ensemble de données PhysioBank. Ce modèle a été construit à partir de zéro sur le concept de confidentialité par conception afin d'être compatible avec 2PC. La solution est implémentée avec le framework ABY [42] qui a nécessité la troncature des valeurs d'entrée et des paramètres du modèle. La deuxième méthode de troncature combinée aux circuits arithmétiques consiste à multiplier les valeurs d'entrée par 10^3 et montre une amélioration significative en termes de performances et de précision. PAC atteint une précision de 96,34%, et les résultats expérimentaux montrent que la prédiction d'un battement cardiaque prend environ 1 s dans des scénarios réels. Nous montrons que plus d'économies peuvent être réalisées avec l'utilisation de lots de pulsations pour effectuer des prédictions (voir le Chapitre 4).

2. SwaNN, comme le montre la Figure A.1, est une classification de réseau de neurones préservant la confidentialité combinant le schéma de cryptage Paillier à homomorphie additive [9] avec 2PC. Grâce à l'utilisation de l'algorithme de chiffrement de Paillier

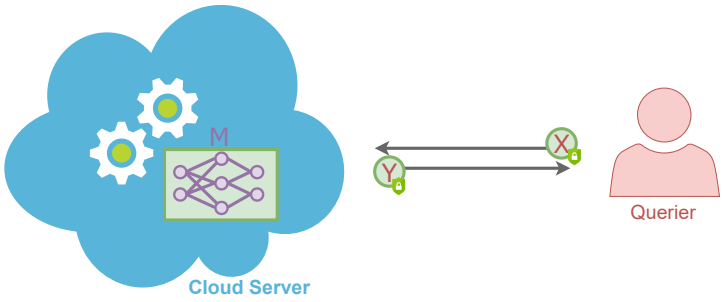


Figure A.1 – Modèle de texte en clair dans le scénario du demandeur et du serveur cloud

pour les opérations linéaires et également de la fonction d'activation x^2, la solution atteint un meilleur coût de calcul par rapport aux solutions existantes à base de (F)HE. Différentes optimisations de calcul basées sur l'utilisation du datapackage et de l'algorithme de multi-exponentiation ont été implémentées. De plus, le coût de communication est également minimisé puisque 2PC n'est utilisé que pour des opérations non linéaires (le pooling `Max` et/ou `RELU`). Les résultats expérimentaux montrent que SwaNN est meilleur en termes de coût de calcul par rapport aux solutions basées sur (F)HE et meilleur en coût de communication par rapport aux solutions basées sur 2PC (voir le Chapitre 4).

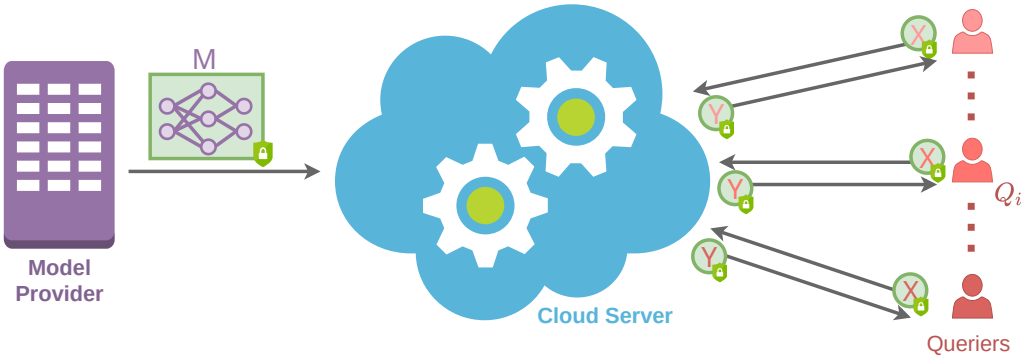


Figure A.2 – Modèle crypté dans le scénario du demandeur et du serveur cloud

3. ProteiNN, comme le montre la Figure A.2, est une solution de classification de réseau neuronal préservant la confidentialité basée sur l'utilisation du recryptage homomorphique du proxy (H-PRE) et du cryptage additif simple. ProteiNN assure la confidentialité du ou des modèles, des entrées et des résultats. De plus, le fournisseur de modèle contrôle également le modèle sous-traité au serveur cloud. Nous avons implémenté ProteiNN comme preuve de concept avec une étude de cas, et notre travail montre des résultats de performance prometteurs (voir le Chapitre 4).

4. pp-TRACLUS, comme le montre la Figure A.3, est la première solution de clustering

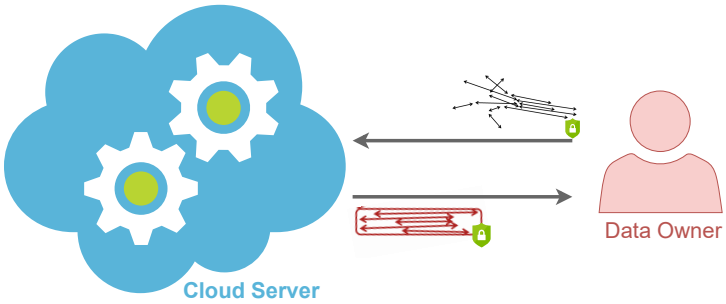


Figure A.3 – Clustering dans le scénario du propriétaire des données et du serveur cloud

de trajectoires préservant la confidentialité basée sur 2PC. Nous avons conçu un protocole efficace pour TRACLUS et l'avons appliqué sur plusieurs jeux de données, y compris un jeu de données du monde réel, à savoir le jeu de données Travel contenant les mouvements de personnes. Nous avons proposé une mesure de distance approchée compatible 2PC pour les trajectoires et évalué sa qualité en montrant qu'elle peut même offrir une meilleure qualité de clustering que la distance TRACLUS d'origine (voir le Chapitre 5).

Dans la deuxième partie de cette thèse, nous étudions un scénario composé de deux serveurs cloud sans collusion (appelés deux serveurs) qui aident le(s) client(s) ou propriétaire(s) des données à exécuter une classification de réseau neuronal préservant la confidentialité, un regroupement de trajectoires., et des tâches d'agrégation de données sans obtenir ou divulguer aucune information concernant les données traitées ou leur sortie. Nous présentons quatre solutions dans cette partie, à savoir:

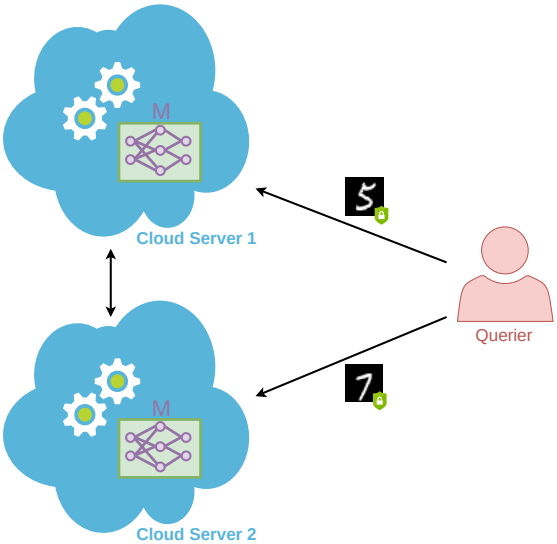


Figure A.4 – Scénario à deux serveurs avec deux images d'entrée dans SwaNN

1. SwaNN avec deux serveurs, comme le montre la Figure A.4, peut être exécuté au cas où le demandeur manque de ressources. Une solution hybride, SwaNN, atteint le meilleur des deux mondes, à savoir une meilleure surcharge de calcul par rapport aux solutions basées sur (F)HE et une meilleure surcharge de communication par rapport aux solutions basées sur 2PC (voir le Chapitre 6).

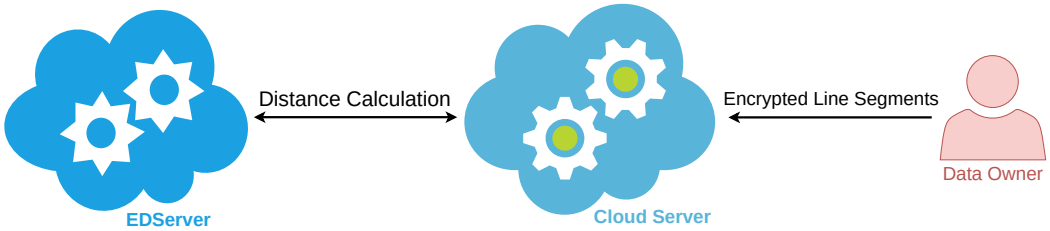


Figure A.5 – pp-TRACLUS basé sur le cryptosystème Paillier

2. pp-TRACLUS, comme le montre la Figure A.5, est un clustering de trajectoires préservant la confidentialité combinant le schéma de cryptage de Paillier additif homomorphe [9] et TRACLUS. Nous proposons d'utiliser la distance euclidienne au carré comme métrique de distance au lieu de la métrique de distance complexe TRACLUS ; par conséquent, la nouvelle métrique est facilement compatible avec le cryptosystème Paillier. De plus, nous proposons un protocole de multiplication sécurisé pour calculer le carré de la distance euclidienne sans divulguer aucune information concernant les segments de ligne (voir le Chapitre 7).

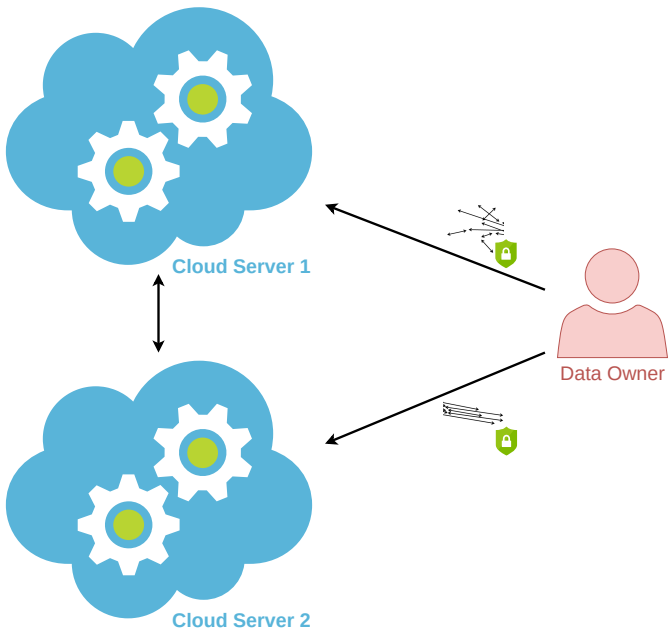


Figure A.6 – Clustering dans le scénario à deux serveurs

3. pp-TRACLUS basé sur 2PC, comme le montre la Figure A.6, est proposé car le schéma de cryptage Paillier encourt des coûts de calcul élevés. Dans cette solution, nous utilisons deux serveurs pour obtenir un protocole TRACLUS préservant la confidentialité plus efficace et réduire la charge de travail du propriétaire des données dans la solution basée sur 2PC avec un seul serveur (voir le Chapitre 7).

4. PRIDA, comme le montre la Figure A.7, est une solution d'agrégation de données préservant la confidentialité qui combine FHE multi-clés avec 2PC et seuil FHE avec 2PC. Grâce à l'utilisation de ces deux blocs de construction cryptographiques avec un paramètre impliquant deux serveurs cloud non-collaboratifs (nous les appelons Agrégateurs dans cette solution), PRIDA prend en charge les scénarios avec plus d'un analyseur de données. De plus, PRIDA permet aux propriétaires de données d'avoir un certain contrôle sur l'analyseur de données qui peut avoir accès aux informations agrégées résultantes. De plus, avec l'introduction d'une phase de comptage anonyme, les analyseurs de données ne peuvent découvrir le résultat de l'agrégation que lorsqu'un nombre suffisant de propriétaires de données (supérieur à un seuil prédéfini) les autorise. Nos résultats expérimentaux semblent prometteurs en termes d'évolutivité (voir Chapitre 8).

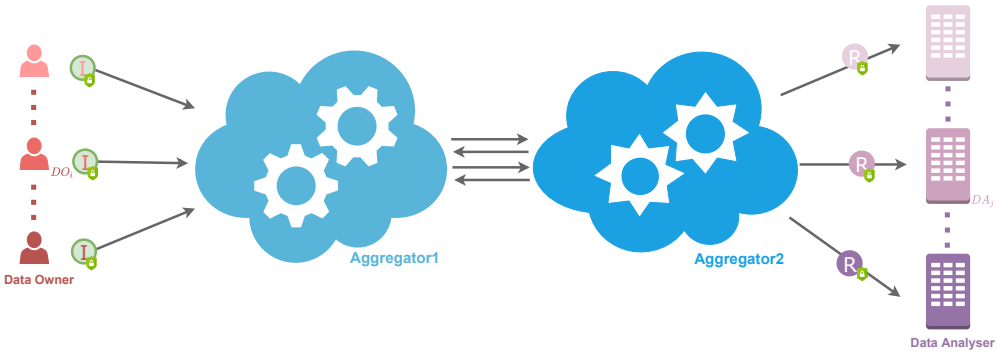


Figure A.7 – PRIDA - Joueurs

Parallèlement à ces chapitres, nous utilisons deux graphiques dans les Figures A.8 et A.9 qui localisent les protocoles de préservation de la confidentialité proposés pour la classification des réseaux neuronaux, TRACLUS et l'agrégation de données concernant leur confidentialité et évaluation de l'exactitude/qualité, et leurs niveaux de confidentialité et d'efficacité.

Pour résumer, nous pensons que nos solutions proposées répondent au besoin de protocoles de préservation de la confidentialité pour les techniques d'apprentissage automatique, permettent la confidentialité des données avec le modèle de thread réaliste et atteignent un bon équilibre entre confidentialité, efficacité et évaluation de la précision/qualité.

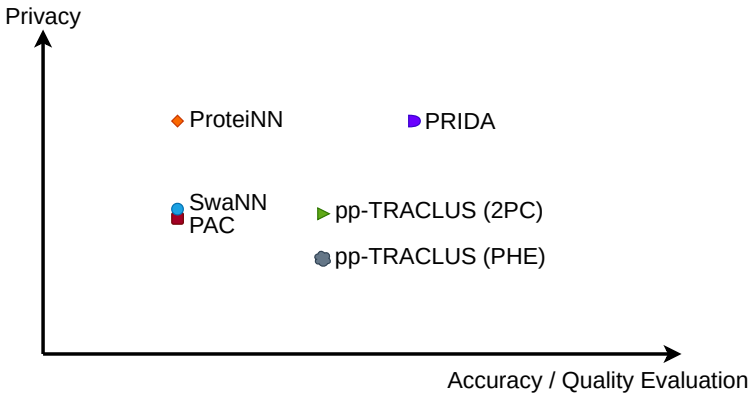


Figure A.8 – Toutes les techniques d'apprentissage automatique préservant la confidentialité proposées par cette thèse concernant le compromis entre la confidentialité et l'évaluation de la précision/de la qualité.

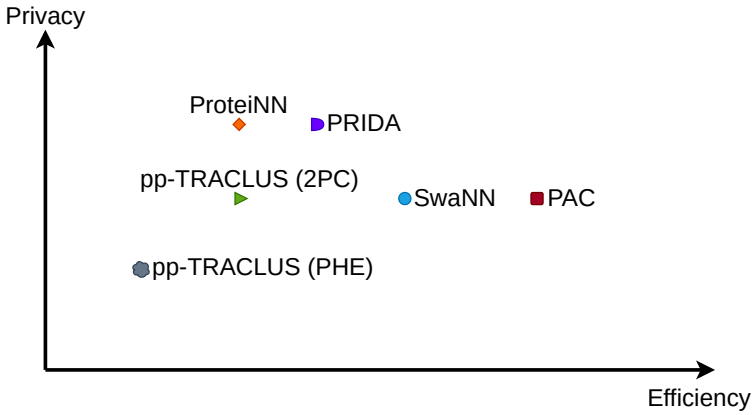


Figure A.9 – Toutes les techniques d'apprentissage automatique préservant la confidentialité proposées par cette thèse concernant le compromis entre confidentialité et efficacité.

Bibliography

[1] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, "Mlaas: Machine learning as a service," in *14th IEEE International Conference on Machine Learning and Applications, ICMLA 2015, Miami, FL, USA, December 9-11*, 2015.

[2] T. E. Parliment and the Council of the European Union, "European General Data Protection Regulation," https://gdpr.eu/, 2018.

[3] T. E. Commission, "Regulation of the european parliament and of the council concerning the respect for private life and the protection of personal data in electronic communications and repealing directive 2002/58/ec (regulation on privacy and electronic communications)," 2016.

[4] N. Jones, "Computer science: The learning machines," *Nature*, 2014.

[5] E. Alpaydin, *Introduction to Machine Learning.* The MIT Press, 2010.

[6] S. Haykin, *Neural Networks and Learning Machines.* Pearson Education, 2011.

[7] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: A partition-and-group framework," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, 2007.

[8] M. Cunguara, T. Silva, and P. Pedreiras, "Multipath data aggregation on wsn," in *IEEE + ACM Workshop on Signal Processing Advances in Sensor Networks (part of CPSWeek) - SPADES-Nets*, April 2013.

[9] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, 1999.

[10] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce english text," *Complex Systems*, vol. 1, 1987.

[11] P. J. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," in *General Systems*, vol. XXII, 1977, pp. 25–38.

[12] K. Labusch, E. Barth, and T. Martinetz, "Simple method for high-performance digit recognition based on sparse coding," *Neural Networks, IEEE Transactions on*, vol. 19, no. 11, pp. 1985–1989, Nov. 2008.

[13] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[14] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[15] V. Vovk, "Kernel ridge regression," in *Empirical inference*. Springer, 2013, pp. 105–116.

[16] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 20, no. 2, pp. 215–232, 1958.

[17] W. Light, "Ridge functions, sigmoidal functions and neural networks," in *In Approximation Theory VII*. Academic Press, 1993, pp. 163–206.

[18] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[19] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Systems, Man, and Cybernetics*, 1983.

[20] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *CoRR*, 2012.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, 2012.

[22] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12*, 2015.

[23] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[24] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96, 1996.

[25] A. Singh, A. Yadav, and A. Rana, "K-means with three different distance metrics," *International Journal of Computer Applications*, vol. 67, 2013.

[26] P. D. Grünwald, J. I. Myung, and M. A. Pitt, *Advances in Minimum Description Length: Theory and Applications*. Cambridge, Massachusetts: MIT Press, 2005.

[27] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," in *SCIENCE*, 1983.

[28] P. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis," in *Journal of Computational and Applied Mathematics*, 1987.

[29] H.-P. Kriegel and M. Pfeifle, "Density-Based Clustering of Uncertain Data," in *SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2005.

[30] D. Moulavi, P. A. Jaskowiak, R. J. G. B. Campello, A. Zimek, and J. Sander, "Density-based Clustering Validation," in *International Conference on Data Mining*. SIAM, 2014.

[31] Y. Lindell, "How to simulate it - A tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography*, 2017.

[32] ——, "Secure Multiparty Computation (MPC)," in *Cryptology ePrint Archive, Report 2020/300*, 2020.

[33] M. Bellare, D. Pointcheval, and P. Rogaway, "Relations among notions of security for public-key encryption schemes," in *CRYPTO*, 1998.

[34] A. C. Yao, "Protocols for secure computations (extended abstract)," in *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, 1982.

[35] A. C.-C. Yao, "How to generate and exchange secrets," in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, 1986.

[36] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, 1987.

[37] D. Ikarashi, R. Kikuchi, K. Hamada, and K. Chida, "Actively private and correct mpc scheme in $t < n/2$ from passively secure schemes with small overhead," Cryptology ePrint Archive, Report 2014/304, 2014, https://eprint.iacr.org/2014/304.

[38] M. O. Rabin, "How to exchange secrets with oblivious transfer," Cryptology ePrint Archive, Report 2005/187, 2005.

[39] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, 1991.

[40] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay—a secure two-party computation system," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. USA: USENIX Association, 2004, p. 20.

[41] A. Ben-David, N. Nisan, and B. Pinkas, "Fairplaymp: A system for secure multi-party computation," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 257–266.

[42] D. Demmler, T. Schneider, and M. Zohner, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.

[43] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," Cryptology ePrint Archive, Report 2020/521, 2020, https://eprint.iacr.org/2020/521.

[44] L. Braun, D. Demmler, T. Schneider, and O. Tkachenko, "Motion - a framework for mixed-protocol multi-party computation," Cryptology ePrint Archive, Report 2020/1137, 2020, https://eprint.iacr.org/2020/1137.

[45] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, Jul. 2018.

[46] C. Fontaine and F. Galand, "A survey of homomorphic encryption for non-specialists," *EURASIP Journal on Information Security*, p. Article ID 13801, 2007. [Online]. Available: https://hal.inria.fr/inria-00504233

[47] N. P. Smart, *Cryptography Made Simple*, 1st ed. Springer Publishing Company, Incorporated, 2015.

[48] R. Rivest, L. Adleman, and M. Dertouzos, "On data banks and privacy homomorphisms," in *Foundations on Secure Computation, Academia Press*, 1978.

[49] C. Gentry, "A Fully Homomorphic Encryption Scheme," 2009.

[50] ——, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC*, 2009.

[51] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, "A guide to fully homomorphic encryption," Cryptology ePrint Archive, Report 2015/1192, 2015, https://ia.cr/2015/1192.

[52] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of CRYPTO 84 on Advances in Cryptology*, 1985.

[53] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, 1978.

[54] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, 2001.

[55] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," in *Proceedings of the Second International Conference on Theory of Cryptography*, 2005.

[56] M. Fellows and N. Koblitz, "Combinatorial cryptosystems galore!" *Contemp. Math*, 1994.

[57] M. Albrecht, P. Farshim, J.-C. Faugere, and L. Perret, "Polly cracker, revisited," *Advances in Cryptology–ASIACRYPT*, 2011.

[58] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Des. Codes Cryptography*, 2014.

[59] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS)*, 2012.

[60] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Report 2012/144, 2012.

[61] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," 2013.

[62] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," Cryptology ePrint Archive, Report 2016/421, 2016.

[63] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Advances in Cryptology - ASIACRYPT - 22nd International Conference on the Theory and Application of Cryptology and Information Security*, 2016.

[64] ——, "Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE," in *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, 2017.

[65] J. H. Cheon and D. Stehlé, "Fully homomophic encryption over the integers revisited," in *Advances in Cryptology – EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds., 2015.

[66] Z. Brakerski, "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP," *CRYPTO*, 2012. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-32009-5_50

[67] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachéne, "Tfhe: Fast fully homomorphic encryption over the torus," Cryptology ePrint Archive, Report 2018/421, 2018.

[68] M. Blaze and M. Strauss, "Atomic proxy cryptography," Proc. EuroCrypt '97, Tech. Rep., 1998.

[69] A.-A. Ivan and Y. Dodis, "Proxy cryptography revisited," in *NDSS*, 2003.

[70] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM Conference on Computer and Communications Security*, 1993.

[71] ——, "The Exact Security of Digital Signatures-how to Sign with RSA and Rabin," in *Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques*, 1996.

[72] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *In EUROCRYPT*. Springer-Verlag, 1998, pp. 127–144.

[73] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Trans. Inf. Syst. Secur.*, 2006.

[74] D. Boneh, "The decision diffie-hellman problem," in *Proceedings of the Third International Symposium on Algorithmic Number Theory*, ser. ANTS-III, 1998.

[75] R. Canetti and S. Hohenberger, "Chosen-ciphertext secure proxy re-encryption," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[76] C. Ma, J. Li, and W. Ouyang, "A homomorphic proxy re-encryption fromźlattices," in *Proceedings of the 10th International Conference on Provable Security - Volume 10005*, 2016.

[77] R. Bellafqira, G. Coatrieux, D. Bouslimi, G. Quellec, and M. Cozic, "Sharing data homomorphically encrypted with different encryption keys," *CoRR*, 2017.

[78] D. Derler, S. Ramacher, and D. Slamanig, "Homomorphic proxy re-authenticators and applications to verifiable multi-user data aggregation," Cryptology ePrint Archive, Report 2017/086, 2017.

[79] W. Ding, Z. Yan, and R. H. Deng, "Encrypted data processing with homomorphic re-encryption," *Inf. Sci.*, 2017.

[80] Y. Polyakov, K. Rohloff, G. Sahu, and V. Vaikuntanathan, "Fast proxy re-encryption for publish/subscribe systems," *ACM Trans. Priv. Secur.*, 2017.

[81] R. Bellafqira, G. Coatrieux, D. Bouslimi, G. Quellec, and M. Cozic, "Proxy re-encryption based on homomorphic encryption," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017.

[82] D. Derler, S. Krenn, T. Lorünser, S. Ramacher, D. Slamanig, and C. Striecks, "Revisiting proxy re-encryption: Forward secrecy, improved security, and applications," Cryptology ePrint Archive, Report 2018/321, 2018.

[83] H. S., "Homomorphic encryption," *Tutorials on the Foundations of Cryptography, Information Security and Cryptography*, 2017.

[84] PALISADE, "PALISADE Lattice Cryptography Library (release 1.10.6)," https://palisade-crypto.org/, Dec. 2020.

[85] A. Lopez-Alt, E. Tromer, and V. Vaikuntanathan, "Cloud-assisted multiparty computation from fully homomorphic encryption," Cryptology ePrint Archive, Report 2011/663, 2011, https://eprint.iacr.org/2011/663.

[86] ——, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," Cryptology ePrint Archive, Report 2013/094, 2013, https://eprint.iacr.org/2013/094.

[87] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference," *CCS*, 2019.

[88] H. Chen, I. Chillotti, and Y. Song, "Multi-Key Homomophic Encryption from TFHE," Cryptology ePrint Archive, Report 2019/116, 2019.

[89] ——, "MK-TFHE Library," https://github.com/ilachill/MK-TFHE, 2019.

[90] G. Asharov, A. Jain, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold FHE," Cryptology ePrint Archive, 2011. [Online]. Available: https://eprint.iacr.org/2011/613

[91] A. Lopez-Alt, E. Tromer, and V. Vaikuntanathan, "Cloud-assisted multiparty computation from fully homomorphic encryption," Cryptology ePrint Archive, Report 2011/663, 2011, https://eprint.iacr.org/2011/663.

[92] Y. Lindell, "Secure Multiparty Computation for Privacy-Preserving Data Mining," 2008.

[93] M. Mansouri, B. Bozdemir, M. Önen, and O. Ermis, "PAC: Privacy-preserving arrhythmia classification with neural networks," in *FPS 2019, 12th International Symposium on Foundations and Practice of Security*, 2019.

[94] G. Tillem, B. Bozdemir, and M. Önen, "SwaNN: Switching among cryptographic tools for privacy-preserving neural network predictions," in *SECRYPT 2020, 17th International Conference on Security and Cryptography, 8-10 July 2020, Lieusaint-Paris, France*, 2020. [Online]. Available: https://www.eurecom.fr/~{}bozdemir/Swannfull.pdf

[95] B. Bozdemir, O. Ermis, and M. Önen, "ProteiNN: Privacy-preserving one-to-many neural network classifications," in *SECRYPT 2020, 17th International Joint Conference on Security and Cryptography, 8-10 July 2020, Lieusaint-Paris, France*, 2020.

[96] M. Azraoui, M. Bahram, B. Bozdemir, S. Canard, E. Ciceri, O. Ermis, R. Masalha, M. Mosconi, M. Önen, M. Paindavoine, B. Rozenberg, B. Vialla, and S. Vicini, "SoK: Cryptography for neural networks," in *In IFIP Summer School on Privacy and Identity Management*, 2019. [Online]. Available: https://www.ifip-summerschool.org/

[97] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03*, 2017.

[98] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: scalable provably-secure deep learning," in *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, 2018.

[99] M. Ball, B. Carmer, T. Malkin, M. Rosulek, and N. Schimanski, "Garbled neural networks are practical," Cryptology ePrint Archive, Report 2019/338, 2019, https://eprint.iacr.org/2019/338.

[100] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08*, 2018.

[101] M. Dahl, J. Mancuso, Y. Dupis, B. Decoste, M. Giraud, I. Livingstone, J. Patriquin, and G. Uhma, "Private machine learning in tensorflow using secure computation," *CoRR*, 2018.

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

systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[103] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "EzPC: Programmable, efficient, and scalable secure two-party computation for machine learning," in *(IEEE EuroS&P 2019)*, 2019.

[104] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: Xnor-based oblivious deep neural network inference," in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, Aug. 2019.

[105] A. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," in *Proceedings on Privacy Enhancing Technologies*, 2020.

[106] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," Cryptology ePrint Archive, Report 2020/1002, 2020.

[107] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 2016.

[108] Microsoft Research, "Microsoft SEAL (release 3.6)," https://github.com/Microsoft/SEAL, Nov. 2020.

[109] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *CoRR*, 2017.

[110] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *IACR Cryptology ePrint Archive*, 2017.

[111] A. Ibarrondo and M. Önen, "Fhe-compatible batch normalization for privacy preserving deep learning," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings*, 2018.

[112] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, 2018.

[113] A. Sanyal, M. J. Kusner, A. Gascón, and V. Kanade, "TAPAS: tricks to accelerate (encrypted) prediction as a service," *CoRR*, 2018.

[114] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *PoPETs*, 2018.

[115] X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19*, 2018.

[116] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *CoRR*, 2018.

[117] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "CHET: an optimizing compiler for fully homomorphic neural-network inferencing," in *The 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020.

[118] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data," In Proceedings of the 16th ACM International Conference on Computing Frontiers, 2019.

[119] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "nGraph-HE2: A high-throughput framework for neural network inference on encrypted data," In Proceedings of the 7th ACM Workshop on Encrypted Computing and Applied Homomorphic Cryptography, 2019.

[120] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: a mixed protocol machine learning framework for private inference," In Proceedings of the 15th International Conference on Availability, Reliability and Security, 2020.

[121] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *Proceedings of the 8th workshop on Multimedia & Security, MM&Sec, Geneva, Switzerland, September 26-27, 2006*, 2006.

[122] C. Orlandi, A. Piva, and M. Barni, "Oblivious neural network computing via homomorphic encryption," *EURASIP J. Information Security*, 2007.

[123] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, 2018.

[124] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "DELPHI: A cryptographic inference service for neural networks," in *29th USENIX Security Symposium (USENIX Security 19)*, 2020.

[125] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[126] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *CoRR*, 2018.

[127] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[128] J. Cabrero-Holgueras and S. Pastrana, "Sok: Privacy-preserving computation techniques for deep learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 4, pp. 139–162, 2021.

[129] R. E. Kass and C. E. Clancy, "Basis and Treatment of Cardiac Arrhythmias," 2006.

[130] I. T. Jolliffe, "Principal Component Analysis," 2002.

[131] F. Castells, P. Laguna, L. Sörnmo, A. Bollmann, and J. M. Roig, "Principal Component Analysis in ECG Signal Processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 74580, 2007.

[132] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, 1987.

[133] M. Mansouri, B. Bozdemir, M. Önen, and O. Ermis, "PAC: Privacy-preserving arrhythmia classification with neural networks," in *Technical Report*, 2019. [Online]. Available: http://www.eurecom.fr/fr/publication/5998/download/sec-publi-5998.pdf

[134] T. Toft, "Sub-linear, secure comparison with two non-colluding parties," in *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*, 2011.

[135] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, 2017.

[136] W. Henecka, S. Kögl, A. Sadeghi, T. Schneider, and I. Wehrenberg, "TASTY: tool for automating secure two-party computations," in *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8*, 2010.

[137] T. Bianchi, A. Piva, and M. Barni, "Composite signal representation for fast and storage-efficient processing of encrypted signals," *IEEE Trans. Information Forensics and Security*, 2010.

[138] C. H. Lim and P. J. Lee, "More flexible exponentiation with precomputation," in *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, 1994.

[139] C. H. Lim, "Efficient multi-exponentiation and application to batch verification of digital signatures," *Unpublished manuscript, August*, 2000.

[140] S. Halevi, Y. Polyakov, and V. Shoup, "An improved rns variant of the bfv homomorphic encryption scheme," Cryptology ePrint Archive, 2018.

[141] M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Security of homomorphic encryption," HomomorphicEncryption.org, Tech. Rep., 2017.

[142] S. Samet, A. Miri, and L. Orozco-Barbosa, "Privacy preserving k-means clustering in multi-party environment." in *SECRYPT*, 2007.

[143] S. Jha, L. Kruger, and P. McDaniel, "Privacy preserving clustering," in *Proceedings of the 10th European Conference on Research in Computer Security*, ser. ESORICS'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 397–417.

[144] G. Jagannathan, K. Pillaipakkamnatt, and R. N. Wright, "A new privacy-preserving distributed k-clustering algorithm," in *SDM*, 2006.

[145] A. Jäschke and F. Armknecht, "Unsupervised machine learning on encrypted data," *IACR Cryptology ePrint Archive*, vol. 2018, p. 411, 2018.

[146] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, ser. KDD '05. New York, NY, USA: ACM, 2005, pp. 593–599. [Online]. Available: http://doi.acm.org/10.1145/1081870.1081942

[147] P. Bunn and R. Ostrovsky, "Secure two-party k-means clustering," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[148] D. Liu, E. Bertino, and X. Yi, "Privacy of outsourced k-means clustering," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '14. New York, NY, USA: ACM, 2014, pp. 123–134. [Online]. Available: http://doi.acm.org/10.1145/2590296.2590332

[149] I. V. Anikin and R. M. Gazimov, "Privacy preserving dbscan clustering algorithm for vertically partitioned data in distributed systems," in *2017 International Siberian Conference on Control and Communications (SIBCON)*, June 2017, pp. 1–4.

[150] P. Mohassel, M. Rosulek, and N. Trieu, "Practical Privacy-Preserving K-means Clustering," in *PoPETS*. Sciendo, 2020.

[151] K. A. Kumar and C. P. Rangan, "Privacy preserving dbscan algorithm for clustering," in *Advanced Data Mining and Applications*, R. Alhajj, H. Gao, J. Li, X. Li, and O. R. Zaïane, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 57–68.

[152] X. Wei-jiang, H. Liu-sheng, L. Yong-long, Y. Yi-fei, and J. Wei-wei, "Privacy-preserving DBSCAN clustering Over vertically partitioned data," in *International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*. IEEE, 2007, pp. 850–856.

[153] J. Liu, J. Z. Huang, J. Luo, and L. Xiong, "Privacy preserving distributed dbscan clustering," in *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, ser. EDBT-ICDT '12. New York, NY, USA: ACM, 2012, pp. 177–185. [Online]. Available: http://doi.acm.org/10.1145/2320765.2320819

[154] W. M. Wu and H. K. Huang, "A DP-DBScan Clustering Algorithm based on Differential Privacy Preserving," in *Computer Engineering and Science*, 2015.

[155] L. Ni, C. Li, X. Wang, H. Jiang, and J. Yu, "DP-MCDBSCAN: Differential Privacy Preserving Multi-Core DBSCAN Clustering for Network User Data," in *IEEE Access*. IEEE, 2018.

[156] N. Pelekis, A. Gkoulalas-Divanis, M. Vodas, A. Plemenos, D. Kopanaki, and Y. Theodoridis, "Private-HERMES: A Benchmark Framework for Privacy-Preserving Mobility Data Querying and Mining Methods," in *Extending Database Technology*. ACM, 2012.

[157] D. Kopanaki, N. Pelekis, A. Gkoulalas-Divanis, M. Vodas, and Y. Theodoridis, "A Framework for Mobility Pattern Mining and Privacy- Aware Querying of Trajectory Data," in *Hellenic Data Management Symposium*, 2012.

[158] E. Pagnin, G. Gunnarsson, P. Talebi, C. Orlandi, and A. Sabelfeld, "TOPPool: Time-aware Optimized Privacy-Preserving Ridesharing," in *PoPETS*. Sciendo, 2019.

[159] U. M. Aïvodji, K. Huguenin, M. Huguer, and M. Killijian, "Sride: A Privacy-Preserving Ridesharing System," in *WISEC*. ACM, 2018.

[160] P. Hallgren, C. Orlandi, and A. Sabelfeld, "PrivatePool: Privacy-Preserving Ridesharing," in *Computer Security Foundations (CSF)*. IEEE, 2017.

[161] A. Hegde, H. Möllering, T. Schneider, and H. Yalame, "Sok: Efficient privacy-preserving clustering," Cryptology ePrint Archive, Report 2021/809, 2021, https://ia.cr/2021/809.

[162] B. Bozdemir, S. Canard, O. Ermis, H. Möllering, M. Önen, and T. Schneider, "Privacy-preserving density-based clustering," in *ACM ASIACCS 2021, 16th ACM ASIA Conference on Computer and Communications Security*, 2021.

[163] Y. Zheng, "Location-based social networks: Users," Computing with Spatial Trajectories, Springer, 2011, https://doi.org/10.1007/978-1-4614-1629-6_8.

[164] P. Besse, B. Guillouet, J.-M. Loubes, and F. Royer, "Review & Perspective for Distance Based Clustering of Vehicle Trajectories," in *Transactions on Intelligent Transportation Systems*. IEEE, 2016.

[165] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Privacy Enhancing Technologies*, I. Goldberg and M. J. Atallah, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 235–253.

[166] P. Mohassel and P. Rindal, "Aby^3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19*, 2018.

[167] S. Wagh, D. Gupta, and N. Chandran, "Securenn: Efficient and private neural network training," in *Privacy Enhancing Technologies Symposium*. (PETS 2019), 2019.

[168] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," Cryptology ePrint Archive, Report 2019/1049, 2019.

[169] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "ASTRA: High throughput 3pc over rings with application to secure prediction," The 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, 2019.

[170] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "FLASH: fast and robust framework for privacy-preserving machine learning," Proceedings on Privacy Enhancing Technologies, 2020.

[171] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4pc framework for privacy preserving machine learning," Cryptology ePrint Archive, Report 2019/1315, 2019.

[172] A. Ibarrondo, H. Chabanne, and M. Önen, "Banners: Binarized neural networks with replicated secret sharing," Cryptology ePrint Archive, 2021.

[173] J. Vaidya and C. Clifton, "Privacy-preserving k-means clustering over vertically partitioned data," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 206–215. [Online]. Available: http://doi.acm.org/10.1145/956750.956776

[174] M. C. Doganay, T. B. Pedersen, Y. Saygin, E. Savaş, and A. Levi, "Distributed privacy preserving k-means clustering with additive secret sharing," in *Proceedings of the 2008 International Workshop on Privacy and Anonymity in Information Society*, 2008.

[175] Z. Gheid and Y. Challal, "Efficient and privacy-preserving k-means clustering for big data mining," in *2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, August 23-26, 2016*, 2016, pp. 791–798. [Online]. Available: https://doi.org/10.1109/TrustCom.2016.0140

[176] W. Du and M. Atallah, "Privacy-preserving cooperative statistical analysis," in *Proceedings of the 17th Annual Computer Security Applications Conference*, ser. ACSAC '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 102–. [Online]. Available: http://dl.acm.org/citation.cfm?id=872016.872181

[177] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk, "Privacy-preserving user clustering in a social network," in *2009 First IEEE International Workshop on Information Forensics and Security (WIFS)*, Dec 2009, pp. 96–100.

[178] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *Proceedings of the 12th Australasian Conference on Information Security and Privacy*, ser. ACISP'07. Berlin, Heidelberg: Springer-Verlag, 2007, p. 416–430.

[179] M. Upmanyu, A. M. Namboodiri, K. Srinathan, and C. V. Jawahar, "Efficient privacy preserving k-means clustering," in *Proceedings of the 2010 Pacific Asia Conference on Intelligence and Security Informatics*, ser. PAISI'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 154–166.

[180] F. Rao, B. K. Samanthula, E. Bertino, X. Yi, and D. Liu, "Privacy-preserving and outsourced multi-user k-means clustering," in *2015 IEEE Conference on Collaboration and Internet Computing (CIC)*, Oct 2015, pp. 80–89.

[181] C. Ding, D. Pei, and A. Salomaa, *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography.* USA: World Scientific Publishing Co., Inc., 1996.

[182] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk, "Privacy-preserving distributed clustering," *EURASIP Journal on Information Security*, vol. 2013, no. 1, p. 4, Nov 2013.

[183] M. Beye, Z. Erkin, and R. L. Lagendijk, "Efficient privacy preserving k-means clustering in a three-party setting," in *2011 IEEE International Workshop on Information Forensics and Security*, Nov 2011, pp. 1–6.

[184] W. Wu, J. Liu, H. Wang, J. Hao, and M. Xian, "Secure and Efficient Outsourced K-means Clustering using Fully Homomorphic Encryption with Ciphertext Packing Technique," in *Transactions on Knowledge and Data Engineering.* IEEE, 2020.

[185] M. Rahman, A. Basu, and S. Kiyomoto, "Towards outsourced privacy-preserving multiparty dbscan," in *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC).* Los Alamitos, CA, USA:

IEEE Computer Society, jan 2017, pp. 225–226. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/PRDC.2017.42

[186] N. Almutairi, F. Coenen, and K. Dures, "Secure third party data clustering using ϕ data: Multi-user order preserving encryption and super secure chain distance matrices," Artificial Intelligence XXXV, 2018, https://www.springerprofessional.de/secure-third-party-data-clustering-using-data-multi-user-order-p/16295950.

[187] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 2010.

[188] E. Shi, T. H. Chan, E. G. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *NDSS*, 2011.

[189] K. Kursawe, G. Danezis, and M. Kohlweiss, "Privacy-friendly aggregation for the smart-grid," *PoPETS*, 2011.

[190] T. H. Chan, E. Shi, and D. Song, "Privacy-preserving stream aggregation with fault tolerance," Cryptology ePrint Archive, Report 2011/655, 2012.

[191] M. Joye and B. Libert, "A scalable scheme for privacy-preserving aggregation of time- series data," *FC*, 2013.

[192] F. Benhamouda, M. Joye, and B. Libert, "A new framework for privacy-preserving aggregation of time-series data," *Association for Computing Machinery*, 2016.

[193] I. Leontiadis, K. Elkhiyaoui, and R. Molva, "Private and dynamic time-series data aggregation with trust relaxation," *CANS*, 2014.

[194] I. Bilogrevic, J. Freudiger, E. D. Cristofaro, and E. Uzun, "What's the gist? privacy-preserving aggregation of user profiles," *ESORICS*, 2014.

[195] N. Papernot, M. Abadi, Ú. Erlingsson, I. J. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," in *ICLR*, 2017.

[196] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Úlfar Erlingsson, "Scalable private learning with pate," in *ICLR*, 2018.

[197] D. Froelicher, P. Egger, J. S. Sousa, J. L. Raisaro, Z. Huang, C. Mouchet, B. Ford, and J.-P. Hubaux, "Unlynx: A decentralized system for privacy-conscious data sharing," *Proceedings on Privacy Enhancing Technologies*, 2017.

[198] D. Froelicher, J. R. Troncoso-Pastoriza, J. S. Sousa, and J. Hubaux, "Drynx: Decentralized, secure, verifiable system for statistical queries and machine learning on distributed datasets," *CoRR*, 2019. [Online]. Available: http://arxiv.org/abs/1902.03785

[199] D. Froelicher, J. R. Troncoso-Pastoriza, A. Pyrgelis, S. Sav, J. S. Sousa, J. Bossuat, and J. Hubaux, "Scalable privacy-preserving distributed learning," *CoRR*, 2020. [Online]. Available: https://arxiv.org/abs/2005.09532

[200] B. Balle, J. Bell, A. Gascón, and K. Nissim, "Private summation in the multi-message shuffle model," *CoRR*, 2020. [Online]. Available: https://arxiv.org/abs/2002.00817

[201] J. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly)logarithmic overhead," Cryptology ePrint Archive, Report 2020/704, 2020.

[202] Z. Erkin, "Private data aggregation with groups for smart grids in a dynamic setting using crt," in *Proceedings of the 2015 IEEE International Workshop on Information Forensics and Security, WIFS*, 2015.

[203] I. Leontiadis, K. Elkhiyaoui, M. Önen, and R. Molva, "PUDA - privacy and unforgeability for data aggregation," in *CANS*, 2015.

[204] M. Ion, B. Kreuter, E. Nergiz, S. Patel, S. Saxena, K. Seth, D. Shanahan, and M. Yung, "Private intersection-sum protocol with applications to attributing aggregate ad conversions," Cryptology ePrint Archive, Report 2017/738, 2017.

[205] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "Sepia: Privacy-preserving aggregation of multi-domain network events and statistics," in *Proceedings of the 19th USENIX Conference on Security*, 2010.

[206] A. Shamir, "How to share a secret," *ACM*, 1979.

[207] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics," *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017.

[208] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *CCS*, 2017.

[209] G. Tsaloli, G. Banegas, and A. Mitrokotsa, "Practical and provably secure distributed aggregation: Verifiable additive homomorphic secret sharing," *Cryptography*, 2020.

[210] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure competitive learning for linear models," in *S&P*, 2019.

[211] C. Beguier and E. W. Tramel, "SAFER: Sparse secure aggregation for federated learning," 2020.

[212] L. Wang, Q. Pang, S. Wang, and D. Song, "F2ed-learning: Good fences make good neighbors," *CoRR*, 2020.

[213] T. D. Nguyen, P. Rieger, H. Y. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, A.-R. Sadeghi, T. Schneider, and S. Zeitouni, "Flguard: Secure and private federated learning," Cryptology ePrint Archive, Report 2021/025, 2021.

[214] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou, "Prio+: Privacy preserving aggregate statistics via boolean shares," Cryptology ePrint Archive, Report 2021/576, 2021.

[215] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. D. Nguyen, P. Rieger, A. R. Sadeghi, T. Schneider, H. Yalame, and S. Zeitouni, "Safelearn: Secure aggregation for private FEderated Learning," Cryptology ePrint Archive, Report 2021/386, 2021.

[216] F. Karakoç, M. Önen, and Z. Bilgin, "Secure aggregation against malicious users," in *SACMAT*, 2021.

[217] F. D. Garcia and B. Jacobs, "Privacy-friendly energy-metering via homomorphic encryption," *International Workshop on Security and Trust Management*, 2010.

[218] S. Erkin and G. Tsudik, "Private computation of spatial and temporal power consumption with smart meters," 2012.

[219] K. Mandal, G. Gong, and C. Liu, "Nike-based fast privacy-preserving high-dimensional data aggregation for mobile devices," *Submitted to IEEE Transaction on dependable and secure computing*, 2018.

[220] B. Bozdemir, F. Germinario, A. Pisani, J. Klemsa, and M. Önen, "PRIDA: PRIvacy-preserving data aggregation with multiple Data Analysers," in *under submisson*, June 2021.

[221] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption security standard," HomomorphicEncryption.org, Toronto, Canada, Tech. Rep., November 2018.

[222] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *Proceedings of the 25th International Conference on Computer Aided Verification*, ser. Lecture Notes in Computer Science, vol. 8044. Springer International Publishing, 2013, pp. 696–701.

[223] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[224] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," 2010. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html