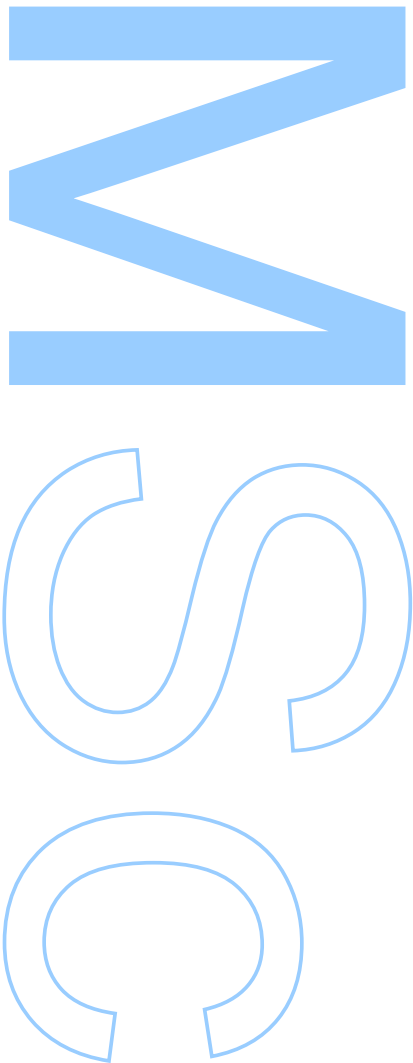
Privacy-Preserving Machine Learning for Network Traffic Analysis

Inês Castro de Macedo
Mestrado em Segurança Informática
Departamento de Ciência de Computadores
2022

Orientador
Prof. Dr. Bernardo Portela, Faculdade de Ciências

Coorientador
Prof. Auxiliar Convidado João Vinagre, Faculdade de Ciências



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

UNIVERSIDADE DO PORTO

MASTERS THESIS

Privacy-Preserving Machine Learning for Network Traffic Analysis

Author:
Inês Castro de MACEDO

Supervisor:
Bernardo PORTELA

Co-supervisor:
João VINAGRE

A thesis submitted in fulfilment of the requirements
for the degree of MSc. Computer Security

at the

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

November 18, 2022

Declaração de Honra

Eu, Inês Castro de Macedo, inscrita no Mestrado em Segurança Informática da Faculdade de Ciências da Universidade do Porto declaro, nos termos do disposto na alínea a) do artigo 14.º do Código Ético de Conduta Académica da U.Porto, que o conteúdo da presente dissertação reflete as perspetivas, o trabalho de investigação e as minhas interpretações no momento da sua entrega.

Ao entregar esta dissertação, declaro, ainda, que a mesma é resultado do meu próprio trabalho de investigação e contém contributos que não foram utilizados previamente noutros trabalhos apresentados a esta ou outra instituição.

Mais declaro que todas as referências a outros autores respeitam escrupulosamente as regras da atribuição, encontrando-se devidamente citadas no corpo do texto e identificadas na secção de referências bibliográficas. Não são divulgados na presente dissertação quaisquer conteúdos cuja reprodução esteja vedada por direitos de autor.

Tenho consciência de que a prática de plágio e auto-plágio constitui um ilícito académico.

Inês Castro de Macedo

Vila Nova de Gaia, 18 de novembro de 2022

Acknowledgements

I would like to thank my parents for their love and support. They have always been there for me in both the good and bad times, and without them, I would not have been able to complete my academic journey.

This work was funded by project THEIA - POCI-01-0247-FEDER-047264, and supported by Fundação para a Ciência e Tecnologia (FCT) under CMU project DAnon - CMU/TIC/0044/2021.

UNIVERSIDADE DO PORTO

Abstract

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

MSc. Computer Security

Privacy-Preserving Machine Learning for Network Traffic Analysis

by Inês Castro de MACEDO

Nowadays, Machine Learning (ML) models are applied in a wide range of areas and industries, such as healthcare, social networks, finances, network traffic analysis, among others. However, due to privacy and security concerns, it is important to keep the ML models and the data used by them secret from one or more parties.

This type of secure ML is known as Privacy-Preserving Machine Learning (PPML) and can be achieved in a variety of ways, such as implementing Secure Multiparty Computation (MPC) protocols. A MPC protocol allows two or more mutually distrustful parties to compute joint functions in a secure manner. Still, the downside of PPML techniques is that they are very computationally expensive, resulting in inefficient practical solutions.

So, in this work, we configured a ML-specific MPC framework, known as TF-Encrypted, to then study two network traffic analysis scenarios, the Torpedo project and the C2 traffic detector. After establishing a baseline for the inference time and the ML classification metrics, we analyzed the performance of both Neural Network (NN) models during a local plaintext computation, a secure MPC computation where all participants run in the same local machine, and a secure distributed computation. Afterwards, we evaluated different NN optimization techniques according to their positive or negative impact on the ML models' performance.

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

Mestrado em Segurança Informática

Machine Learning com Preservação da Privacidade para Análise de Tráfego de Rede

por Inês Castro de MACEDO

Atualmente, os modelos de Machine Learning (ML) são aplicados a uma ampla variedade de áreas e indústrias, como saúde, redes sociais, finanças, análise de tráfego de rede, entre outros. No entanto, devido a questões de privacidade e segurança, é importante manter os modelos de ML e os dados por eles utilizados confidenciais para um ou mais participantes.

Este tipo de ML seguro é conhecido como Machine Learning com Preservação da Privacidade (PPML) e pode ser alcançado de várias maneiras, como implementando protocolos de Computação Colaborativa Segura (MPC). Um protocolo MPC permite que dois ou mais participantes, que não confiam um no outro, calculem funções conjuntas de forma segura. Ainda assim, a desvantagem das técnicas PPML é que elas são muito caras computacionalmente, o que resulta em soluções práticas pouco eficientes.

Então, neste trabalho, configuramos uma framework MPC específica para ML, conhecida como TF-Encrypted, para estudar dois cenários de análise de tráfego de rede, o projeto Torpedo e o detetor de tráfego C2. Após estabelecermos uma baseline para o tempo de inferência e para as métricas de classificação ML, analisamos o desempenho de ambos os modelos de Redes Neuronais (NN) durante uma computação local plaintext, uma computação segura de MPC onde todos os participantes executam na mesma máquina local, e uma computação distribuída segura. De seguida, avaliamos diferentes técnicas de otimização de NN de acordo com o seu impacto positivo ou negativo no desempenho dos modelos de ML.

Contents

List of Figures

List of Tables

Listings

Glossary

C2	Command and Control
CNN	Convolutional Neural Network
FCNN	Fully-Connected Neural Network
GC	Garbled Circuit
ISP	Internet Service Provider
LEA	Law Enforcement Authority
MAC	Message Authentication Code
ML	Machine Learning
MPC	Secure Multiparty Computation
NN	Neural Network
OS	Onion Service
OT	Oblivious Transfer
PPML	Privacy-Preserving Machine Learning
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
VM	Virtual Machine

Chapter 1

Introduction

1.1 Problem Statement

Nowadays, Machine Learning (ML) is applied in a variety of areas, such as healthcare, social networks, finances, as well as network traffic analysis. For example, ML predictors are capable of extremely important tasks, including monitoring chronic diseases [1], detecting credit card fraud [2], studying user's online behavior [3], and even analyzing network traffic flows.

Essentially, ML is an extremely powerful tool that is even making its way into law enforcement to truly enhance its effectiveness during everyday operations. A report published in 2020 by INTERPOL in collaboration with UNICRI includes a series of use cases where ML is applied by law enforcement agencies [4]. For example, the German Central Office for Information Technology in the Security Sector (ZITiS) is working on a recommender system to help during investigations of modern financial crimes, which usually require the study and analysis of large quantities of financial data. This active learning content-based tool will try to interactively query valuable information based on officer's prior searches or similar preferences. A main challenge regarding this tool is making sure that bias does not compromise an investigation by affecting the accuracy and truthfulness of the information. There is also a research center within Japan's National Police Agency (NPA) that is developing three pilot applications to strengthen security surrounding major events. The first tool will be capable of identifying the models of cars in surveillance footage. The second one will analyze suspicious financial transactions that may indicate money laundering. The third one will help identify movements or actions that may be considered suspicious.

However, most of these procedures require handling of private and sensitive data, that if not handled carefully, may lead to the infringement of human rights, as well as data protection laws. So, it is very important to keep both the data and the ML predictor secret to one or more of the parties involved. This secure method of ML is known as Privacy-Preserving Machine Learning (PPML). The main purpose of PPML is to preserve the privacy of data during the process of ML. In other words, the party responsible for performing the necessary ML tasks must not have any knowledge of the data being processed. This in turn requires a mechanism that allows data to be protected, but still be usable by ML. A way to achieve PPML, is to implement a Secure Multiparty Computation (MPC) protocol. For the purposes of this work, we focused on the implementation of a ML-specific Secure Multiparty Computation (MPC) framework, known as TF-Encrypted, to achieve the previous goal.

A MPC protocol allows two or more mutually distrustful parties to compute joint functions in a secure manner. All participants submit their private data and receive the calculated result without learning or having access to any additional information. MPC protocols have more specific security notions, but they guarantee the privacy of all parties and the ML model used during the computation.

Nonetheless, there is a downside to implementing PPML techniques, including MPC protocols. Firstly, even though there have been many advancements over the years, MPC still shows poor scalability to large datasets. Secondly, the computation performed over sensitive data can be quite complex, especially when dealing with Fully-Connected Neural Networks (FCNN) and Convolutional Neural Networks (CNN), which are even more costly than the former. While a deeper NN architecture can lead to more precise and accurate results, it can also lead to memory constraints and inference time bottlenecks. Thirdly, the act of preserving the privacy of the model and the participants' data is also computationally expensive. This means that the combination of these procedures can result in extremely inefficient practical solutions. Therefore, it is critically necessary to optimize secure NNs and increase their range of possible applications.

1.2 Contributions

Considering the resource constraints mentioned before, we studied the theoretical and practical notions of MPC protocols to then apply them to two ML application scenarios related to network traffic analysis. These two scenarios consist of the Torpedo project

[5] and the C2 traffic detector [6][7]. The two scenarios implement a CNN and a FCNN, respectively, to make predictions based on and about traffic flows. We established a baseline for both by analyzing the performance of their respective neural network during a local plaintext computation, a local MPC computation, and a secure distributed computation. Then, we evaluated different optimization techniques and their impact on the NN's overall performance.

1.3 Thesis Structure

The rest of the report is organized as follows:

- Chapter 2 provides the necessary background on secure multiparty computation, cryptographic primitives, security notions, and machine learning classification metrics;
- In chapter 3, we present the state of art of general-purpose and ML-specific MPC protocols, as well as related work;
- Chapter 4 describes the two application scenarios handled during the project, Torpedo and the C2 traffic detector;
- Chapter 5 presents our implementation, optimizations, and subsequent results;
- Finally, chapter 6 describes some possible future work and concludes the report.

Chapter 2

Background

Cryptography is the practice and study of techniques that enable secure communications between a sender and a receiver. These techniques consist of algorithms and protocols based on mathematical concepts capable of protecting information from third parties [8]. Nowadays, cryptography is mainly associated with the process of encryption, where a plaintext is transformed into a ciphertext that can only be decoded by the intended receiver. On the other hand, the process of converting a ciphertext into a plaintext is known as decryption.

But, cryptographic algorithms have many different applications from digital signing, cryptographic keys generation, encrypted web browsing to confidential credit card transactions. No matter its application, cryptography always focuses on providing different security notions, such as confidentiality, integrity, non-repudiation, and authentication. Confidentiality ensures that only the intended receiver and nobody else can access the information [9]. Integrity guarantees that information cannot be modified in storage or when being exchanged between a source and destination, without this alteration being detected [10]. Thirdly, non-repudiation assures that no party cannot deny having sent or received information [11]. Finally, authentication validates the origin and/or destination of a message [12]. These security features are the basis of cryptography.

However, most encryption algorithms only protect information that is either in storage or in transit. They do not protect data that is currently being used or computed. This is due to the fact that information in the first two cases is not being actively changed, it maintains the same value. But, in the third situation, a change to the ciphertext means that the relationship to the plaintext is lost, in other words, the cryptographic mathematical operations would not be able to obtain the correct value of the new plaintext.

Consequently, popular protocols for secure data transmission, such as Advanced Encryption Standard (AES), are completely useless for secure data computations. One way to perform calculations on encrypted data, without needing to decrypt it, is to implement Homomorphic Encryption (HE). Furthermore, the results of the secure computations also remain encrypted. The different types of HE (partially, somewhat, leveled, and fully) represent the computations as either Boolean or arithmetic circuits. While the Fully Homomorphic Encryption (FHE) is capable of potentially evaluating every function through the circuits, the other types of HE have different limitations. Therefore, homomorphic encryption would allow us to protect the confidentiality of machine learning procedures that utilize private and sensitive data. However, there are two major disadvantages, caused by the multiplicative depth of the previously mentioned circuits, that limit the practical applications of HE: large redundancy and high computational complexity [13].

Another way of performing secure computations is to implement Secure Multiparty Computation (MPC). So, in this chapter, we will provide the necessary background on MPC, its cryptographic primitives and security requirements, as well as ML concepts necessary to later test the effectiveness of this computation type using different neural networks. To start, we provide a succinct explanation of MPC, while also presenting the original problem that birthed the concept - Yao's millionaire problem. Secondly, we describe the cryptographic primitives applied by most MPC protocols. Then, we formally define what it means for a MPC protocol to be deemed secure, including its security requirements and its main types of adversaries. Afterwards, we explain the structure and way of working of neural networks, specifically fully-connected neural networks and convolutional neural networks. Finally, we will go over some common ML classification metrics, mainly recall, precision and F1-score.

2.1 Secure Multiparty Computation

Secure Multiparty Computation (MPC) allows mutually distrustful participants to securely compute joint functions by submitting their private data. When the computation is completed, no additional information is revealed beyond the result, even if adversaries collude to obtain more information or alter the result [14]. Essentially, MPC will act as a trusted third-party who accepts each party's private input, computes the requested function and returns the corresponding result. Each party will only have access to their own input–output pair.

As shown in figure 2.1, let us consider a set of N agents and that $f(x_1, x_2, ..., x_N) = (y_1, y_2, ..., y_N)$ is a multivariate function that they wish to compute. For $i \in 1, ..., N$, the MPC service will receive the input x_i from the i-th agent and output the value y_i. However, as previously stated, this is done in a way that no additional information is revealed about the remaining x_j, y_j, for $j \neq i$.

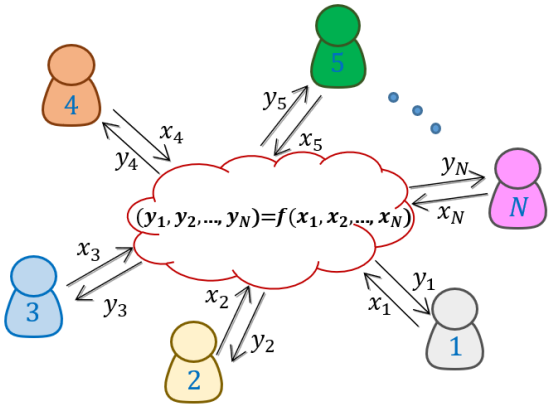


FIGURE 2.1: Example of a secure multiparty computation protocol from [15]

2.1.1 Yao's Millionaire Problem

To better understand the concept of secure multiparty computation, we can also take a look at Andrew Yao's millionaire problem, which originated in 1982. The problem describes a scenario wherein two millionaires, Alice and Bob, wish to know which one of them is richer without revealing the true value of their fortune [16].

Suppose Alice has i millions and Bob has j millions. During the secret computation, it will be determined whether the inequality $i \geq j$ is true or false without revealing the actual values of i and j. In other words, the protocol will determine if Alice is richer than Bob ($i \geq j$), or if Bob is richer than Alice ($i < j$). The information exchanged between the two parties remains secure due to the use of Oblivious Transfer (OT), a cryptographic primitive we will take a look at in the next section.

This scenario is a special case since we can extend the problem to m participants. However, Yao's simple solution was a precedent to the Garbled Circuit (GC) protocol, which is also mentioned in the next section.

2.2 Primitives for Secure Computation

2.2.1 Cryptographic Primitives

This subsection describes the three most important cryptographic primitives applied by most MPC protocols: secret sharing, garbled circuits, and oblivious transfer, which is implemented by garbled circuits. These circuits are a key design characteristic of every major MPC protocol.

Secret sharing is a cryptographic method that takes a secret value, breaks it up into multiple shares and distributes said shares among different parties [14]. Only when the participants join their respective shares can the secret be reconstructed, like shown in figure 2.2. Normally, the dealer, also known as the holder of the secret, creates the shares and sets a threshold t for the number of shares that are required to reconstruct the secret. Any collection with less than t shares reveals nothing about the secret. A secret sharing scheme allows for a more secure storage of highly sensitive data, such as encryption keys.

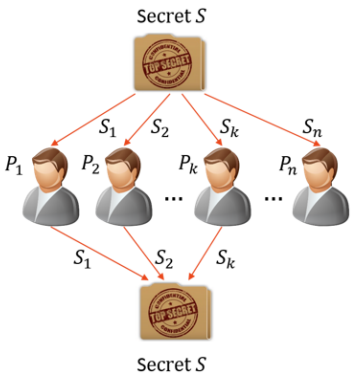


FIGURE 2.2: Secret sharing from [17]

By implementing a secret sharing scheme, a MPC protocol guarantees two security notions: privacy and correctness. These security properties will be addressed in the next subsection.

A **Garbled Circuit** (GC) is the simplest form of MPC involving two parties, where one acts as a garbler and the other performs the role of evaluator [14]. Similarly to Yao's millionaire problem, both participants wish to compute some function without revealing their respective inputs.

Therefore, the garbler will start a special procedure called garbling that encrypts a computation circuit. This circuit is essentially a set of Boolean operations (AND, OR, NOT, XOR) that will be used to evaluate the specified function. In certain cases, the circuits can

also be sets of arithmetic operations (addition, subtraction, multiplication, or division) that will process numbers. Afterwards, the evaluator receives the garbler's encrypted inputs alongside the garbled circuit.

However, to calculate the circuit, the evaluator also needs to garble their input. Since the garbler is the only one capable of encrypting, the evaluator will make use of 1-out-of-2 oblivious transfer to communicate with the garbler and learn the encrypted form of their own inputs, without letting the garbler know which inputs they obtained. During this oblivious transfer protocol, the evaluator acts as the receiver, while the garbler acts as the sender. This protocol is explained in better detail in the next subsection.

Finally, the evaluator decrypts (evaluates) the garbled circuit, obtains the outputs and sends them to the garbler. Figure 2.3 shows a simple example of a Boolean garbled circuit.

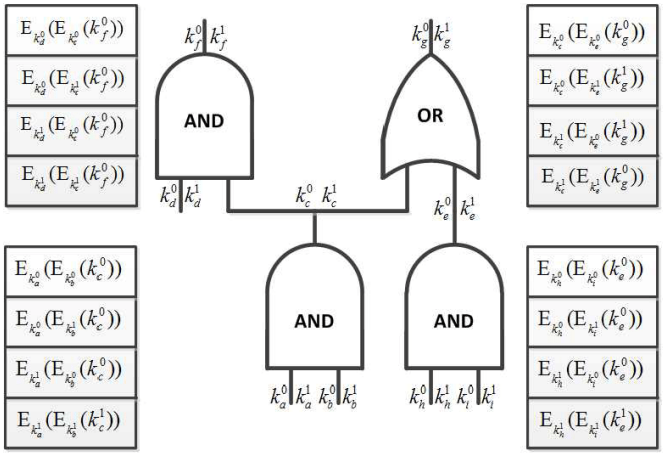


FIGURE 2.3: A simple example of a Boolean garbled circuit from [18]

Oblivious Transfer (OT) is a cryptographic protocol that allows the transfer of selective information between two parties. It allows a participant to choose k of n secrets from another participant, without revealing which secret they chose [14]. The simplest version of the protocol was presented in 1981 by Michael Rabin [19]. During its execution, the sender uses a bit b as its private input and sends it to the receiver with a probability of $\frac{1}{2}$. After the computation, the receiver gets the bit b or an undefined value , while the sender remains oblivious if the message was received or not. There is also 1-out-of-2 OT where a participant chooses as input two bits x_0 and x_1. The other participant then chooses a selection bit b and gets as output the bit x_b. This method is shown in figure 2.4.

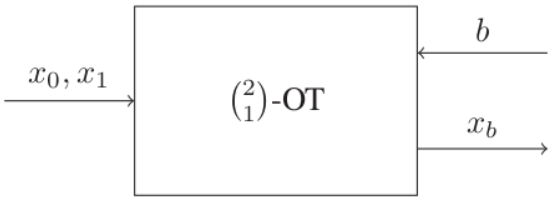


FIGURE 2.4: Oblivious transfer from [14]

2.2.2 Security

Security of a multiparty computation protocol can be formally defined as the comparison of the outcome of a real protocol execution to the outcome of an ideal computation.

In the ideal world, the parties send their inputs to an external trusted and incorruptible party, which then computes the necessary function and sends each party its output. While in the real world, the participants run the protocol amongst themselves without any outside help. So, if the protocol is secure, then no adversary can do more damage in a real execution than in an ideal world execution.

Considering the possibility of adversarial interference, MPC must guarantee certain security requirements alongside the previous definition of security to be deemed secure. All MPC protocols must guarantee privacy and correctness. However, certain protocols can also provide independence of inputs, guaranteed output delivery and fairness. These security notions can be defined as follows:

- **Privacy** ensures that no interested party learns anything beyond what is necessary;
- **Correctness** guarantees that each participant receives its correct output;
- **Independence of inputs** states that corrupted parties must choose their inputs independently of the honest parties' inputs;
- **Guaranteed output delivery** ensures that corrupted parties do not prevent honest participants from receiving their output;
- **Fairness** states that corrupted parties should receive their outputs if and only if the honest parties also receive their outputs.

After establishing the security fundamentals of MPC, we must also consider and describe the main types of possible adversaries, as well as the power they hold. The variables inherent to an attacker include the corruption strategy, the adversarial behavior, the

majority setting, and the computational complexity of the adversary [14], which can be defined as follows:

- **Corruption strategy** answers the question of when and how parties are corrupted. Corruption is a security model mechanism that captures the event when a system is vulnerable to attacks, for example, when a participant decides to act maliciously or when another is hacked. There are three main corruption models:
  - Static corruption model - the adversary possesses a fixed set of parties which it controls. The corrupted parties remain corrupted until the end of the computation;
  - Adaptive corruption model - the adversary is given the capability of corrupting parties during the computation. The corrupted parties remain corrupted until the computation ends;
  - Proactive corruption model - the adversary has the capacity of corrupting parties, however they remain corrupted for a certain period of time, after which they can become honest again.
- **Adversarial behavior** deals with the actions that corrupted participants are allowed to take. There are two main types to consider:
  - Semi-honest adversary - corrupted parties follow the protocol correctly, but their internal state is communicated to the adversary, which can be used to learn information that should have remained private;
  - Malicious adversary - corrupted parties can arbitrarily deviate from the protocol following the adversary's orders. Security against a malicious adversary is implemented as a "malicious-with-abort" scheme, which means that the protocol aborts if malicious activity is detected. While no party will receive an incorrect answer, they may not get no answer at all.
- **Majority setting** depends on the number of corrupt parties involved in the computation. It can be categorized as:
  - Honest majority - the number of corrupt parties, t, is less than half of all participants, n, where $t < n/2$. All security properties are achieved including fairness and guaranteed output delivery;

  - Dishonest majority - the number of corrupt parties, t, is less than the number of all participants, n but greater than half of them, where $n/2 \leq t < n$. However, in this case, all security properties are achieved except for fairness and guaranteed output delivery since the "malicious-with-abort" scheme is in effect.

- **Computational complexity** defines the assumed computational complexity of the adversary. There are two categories:

  - Polynomial-time - the adversary can run in probabilistic polynomial-time. It should be noted that if an attack cannot be carried out in polynomial-time, then it cannot be considered a threat in a real scenario;
  - Computationally unbounded - the adversary has no computational limits.

It is important to note that, usually, assuming weaker adversaries leads to a smaller range of possible attacks. This in turn allows the system to avoid costly operations, such as verifying the integrity of secret shares, which may result in a more efficient system. Therefore, the practicability of MPC depends on the balance between security and performance.

2.3 Machine Learning

2.3.1 Neural Networks

Neural Networks (NN) are a popular choice for classifying and predicting complex data, while maintaining a high level of accuracy when compared to other ML solutions. However, this method requires large amounts of user data, especially personally-identifiable information. As expected, this presents many privacy issues. While users are unable to delete or restrict the use of the already collected information, data owners are sometimes prohibited from publishing or sharing any of their findings due to data protection laws [20]. Therefore, in this work, we will be applying secure multiparty computation to neural networks to achieve a reasonable utility and privacy trade-off. On the one hand, participants will maintain the confidentiality of their data, while respecting the user's privacy. On the other hand, data owners will also be able to share their ML models and respective results with other parties, in a secure manner.

Our goal is to guarantee that the party training or using the ML model cannot obtain any information about it. There is a whole other security branch related to explainability

of the trained model, as well as inference of sensitive information through queries to the model. These are relevant themes, but not directly related to our project, which require an extensive study of ethical artificial intelligence [21]. Even though, we will not be focusing on these themes, they are important to mention considering they are a growing concern related to security.

A Neural Network (NN) is a series of algorithms that try to identify underlying relationships in sets of data by mimicking the way a human brain works. A NN is composed of multiple node layers, including an input layer, one or more hidden layers, and an output layer. While the input layer receives the inputs, the hidden layer will process them and the output layer produces the result. The structure of the different layers that make up a neural network can be seen in figure 2.5.

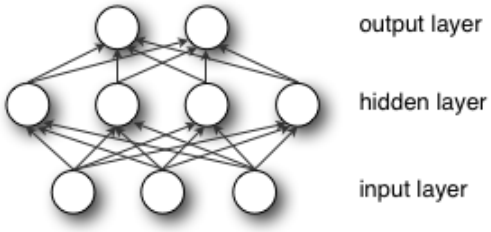


FIGURE 2.5: Diagram of the different node layers within a neural network from [22]

Throughout this learning process, each node acts as an artificial neuron and is connected to another to pass along an activation signal. They also have a weight associated with them that will be adjusted as the learning proceeds. The weight is used to multiply the input received before passing it along to the next layer. Essentially, weights control the strength of the connection between two neurons, which means that a weight decides how much influence the input will have on the output. Figure 2.6 shows a visual representation of a node.

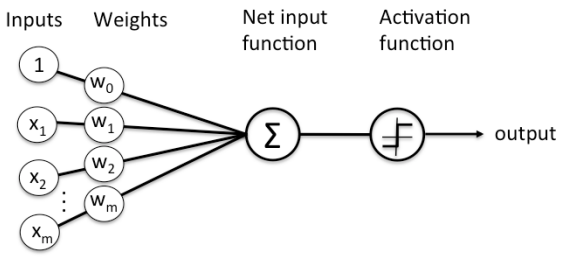


FIGURE 2.6: Diagram of the structure of a neural network node from [22]

NNs are extremely powerful at classifying and clustering data at high velocities, however they first need to be trained with a labeled dataset. Usually, NNs are distinguished by

their depth, meaning the number of node layers through which data must pass to perform pattern recognition [22]. There are many neural network architectures available, but we will be focusing on fully-connected neural networks and convolutional neural networks.

A **Fully-Connected Neural Network** (FCNN) is a type of neural network, where every neuron in one layer connects to every neuron in the next layer. The main advantage of FCNNs is that they are structure agnostic, meaning no special assumptions need to be made about the input [23]. This allows them to be applied to a wide range of problems since they are capable of learning any function necessary. However, they also tend to have weaker performance than other special-purpose networks. Another downside of FCNNs is that they may be prone to overfitting. Overfitting is an obstacle in ML that introduces errors based on noise and meaningless data, which affects the quality of prediction or classification. When dealing with a FCNN, it is important to use a training dataset of sufficient size. Figure 2.7 illustrates a fully-connected neural network with three hidden layers.

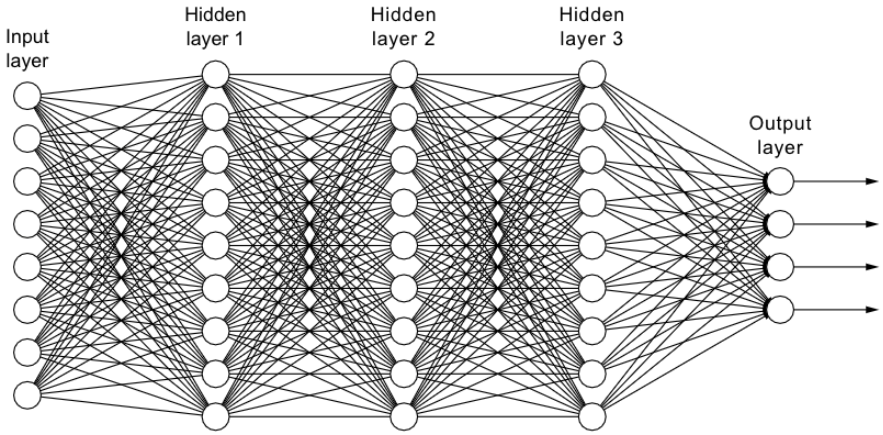


FIGURE 2.7: Simple illustration of a fully-connected neural network from [24]

Nowadays, **Convolutional Neural Networks** (CNN) are one of the most popular neural network architectures. They can be applied in many different areas and domains, but are especially successful at image processing. The current CNNs consist of a lot of node layers, which makes them very deep and a lot more complex than a simple NN. Usually, CNNs implement fully-connected layers near the output layer. Nonetheless, their main building blocks are filters, also known as kernels. These filters are responsible for extracting relevant features from the input data using a convolutional function. A convolution is a mathematical operation that extracts the features from an image that is represented by a matrix in the form of pixels or numbers. Due to this mathematical operation, CNNs

require a lot of processing power, as well as large amounts of data for training. They are also extremely difficult to interpret and configure. A simple illustration of a CNN is shown in figure 2.8.

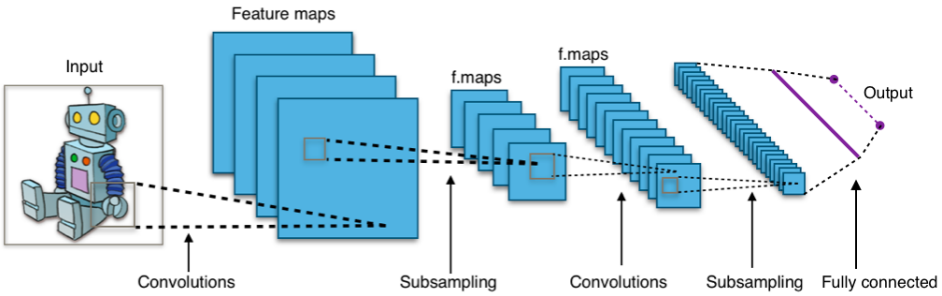


FIGURE 2.8: Simple illustration of a convolutional neural network from [25]

2.3.2 Classification Metrics

The classification metrics are calculated based on the confusion matrix of a ML model. This matrix contains the number of instances of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). TP indicates how many positive class samples the model predicted correctly, while FP shows how many negative class samples the model predicted incorrectly. Finally, TN shows how many negative class samples the model predicted correctly, while FN indicates how many positive class samples the model predicted incorrectly.

The classification metrics are essential to evaluate a model's performance and the overall quality of its classification methods. For this work, we consider the recall, precision and F1-score of each ML model.

Recall calculates the ratio of True Positives to all the positives in ground truth. Mathematically, it is defined as:

$$Recall = \frac{TP}{TP + FN}$$

A model that produces no False Negatives has a recall of 1.0. A higher recall is extremely important in the case of a cancer classifier, since missing even one case of cancer could result in the patient's death.

Precision is the ratio of True Positives and total positives predicted. Mathematically, it can be defined as:

$$Precision = \frac{TP}{TP + FP}$$

A model that produces no False Positives has a precision of 1.0. A higher precision would be more important for a pregnancy classifier, since people might react to a positive result by performing costly decisions like getting married or buying a house.

The **F1-score** combines both precision and recall. It is essentially the harmonic mean of the two metrics. Mathematically, it is defined as:

$$F1 = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

A high F1-score indicates a high precision, as well as high recall values. It symbolizes a good balance between precision and recall.

Chapter 3

State of the Art

As stated in the previous chapter, there are many different ways to perform secure computations with encrypted data, like Homomorphic Encryption (HE), for example. Since this kind of methods can be used to achieve strong security when dealing with ML algorithms, they are all considered Privacy-Preserving Machine Learning (PPML) strategies. The main goal of PPML is to reach a balance between privacy and ML benefits, such as high precision. It allows multiple input sources to train ML models, in a cooperative fashion, without revealing their private data. PPML is extremely useful in today's world, where privacy concerns are ever growing while sensitive data leakages are becoming a regular occurrence [26].

Another common PPML technique is Secure Multiparty Computation (MPC), presented in the previous chapter. While HE's real world applications are still pretty limited, MPC is now such a rapidly changing field that even experts have difficulty keeping track of the different improvements and advancements along the years [14]. So, in this chapter, we study two different categories of MPC protocols: general-purpose and ML-specific protocols.

We will go over the currently available MPC protocols, as well as some related work. Both general-purpose and ML-specific protocols are presented in a succinct manner for an easier comparison between them. These include Sharemind [27], SPDZ [28], SCALE-MAMBA [29], SecureNN [30], FLASH [31], and TF-Encrypted [32]. We will be focusing more on TF-Encrypted since it is the tool we selected to implement the project.

Every framework mentioned here follows the same general approach. First, a compiler will convert a program written in a specific low or high-level language (C, C++, Python) to an intermediate representation, which is often a circuit. Afterwards, the circuit

is passed as input to a runtime that will then execute an MPC protocol and produce an output.

3.1 General-Purpose MPC

General-purpose MPC protocols are capable of securely computing any function, which allows them to be applied to almost every problem and use case [14]. However, they are not the most efficient when compared to special-purpose protocols, such as ML-specific ones. General-purpose compilers are not ready for complex tasks, such as Neural Network (NN) training, which leads to an impossibly long execution time. From the least complex to the most complex, we discuss the inner workings of the Sharemind, SPDZ, and SCALE-MAMBA protocols.

3.1.1 Sharemind

Sharemind [27] securely executes a function using a three-party hybrid protocol, which takes advantage of a custom additive secret sharing scheme. Implementing a hybrid protocol means that Sharemind is not limited to representing computations in the form of circuits. Instead, it can make use of multiple optimized sub-protocols for common operations like comparisons, bit-shifts, and equality tests. These operations are usually expensive to represent with an arithmetic circuit, so by applying the optimized sub-protocols to the secret sharing process, Sharemind can achieve results in a more efficient manner. Meanwhile, the additive secret sharing is a simple secret sharing scheme where k shares are created, and when added together they recreate the original secret.

The three parties involved in the protocol consist of clients who provide inputs, servers who execute the computation, and outputs who receive the final result. The framework was written for exactly three computation servers. However, there is a possibility of having many parties secret sharing their inputs with the three computation servers, which will then save these values in their respective databases for persistent storage. The servers also have a stack to store intermediate results of the different arithmetic operations performed throughout the computation. The deployment diagram of Sharemind can be seen in Figure 3.1.

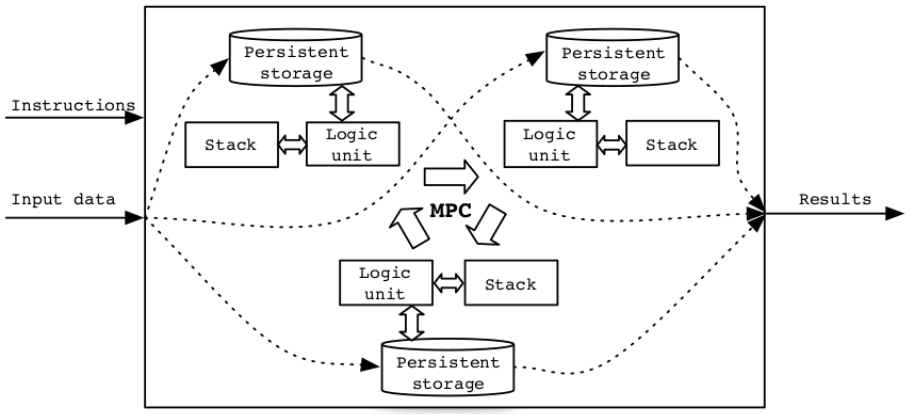


FIGURE 3.1: Deployment diagram of the Sharemind protocol from [27]

In the previous diagram, the outer rectangle encapsulates the MPC process done by the computation servers, which will act as miner nodes. Meanwhile, the input data is provided by different data owners and the instructions are sent to the miners by a data analyst. Each instruction is a command that invokes a secret share computing protocol or to reorder shares. The instructions allow data analysts to build their own specific protocols. When the computation finishes, the miner nodes send the obtained results to the data analyst.

The server code is written in the SecreC language and is then executed using Sharemind's secure runtime. On the other hand, the client and output code utilizes a client library of a common programming language, like C, and is executed using standard compilers [14]. To sum up, the Sharemind protocol can be applied to a wide range of use cases, especially those with particularly large or complex functionalities.

Sharemind's implementation details:

- Development language: C, C++
- Parties supported: 3 parties
- Adversaries tolerated: semi-honest adversaries

3.1.2 SPDZ

SPDZ [28] is the original protocol that led to the development of more optimized versions, such as FRESCO, MP-SPDZ, and SCALE-MAMBA. It allows different participants to securely evaluate arithmetic circuits. It is secure against active and adaptive corruption of up to $n-1$ of the total n players.

SPDZ has two phases: a slower offline pre-processing phase and a faster online phase. In the pre-processing phase, random multiplication triples are created independently of the function to be computed and the secret inputs provided by the participants. To improve the computational efficiency of the MPC protocol, SPDZ also applies Fully or Somewhat Homomorphic Encryption (FHE and SHE) during this phase. Afterwards, in the online phase, when the inputs are finally received, they are then used to compute the specified function [33]. Considering that the pre-processing phase can be executed well in advance and it is the slowest to run out of the two phases, SPDZ provides a low-latency MPC solution. Figure 3.2 displays a high-level view of the SPDZ compiler.

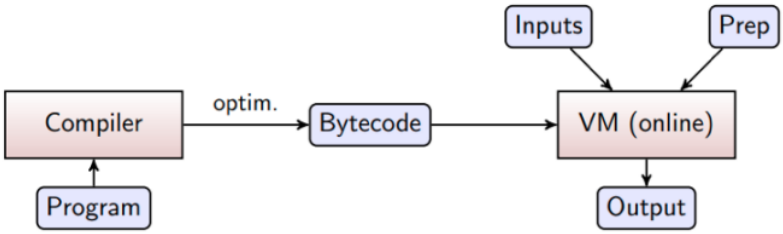


FIGURE 3.2: High-level view of the SPDZ compiler from [34]

This framework added Message Authentication Codes (MAC) to the additive secret sharing process to authenticate values and detect lying adversaries. So, even though the protocol is only robust against passive attacks, it can still be considered secure. Since the corrupt party can only add a known error term to the decryption result, this alteration, that eventually affects the secret shares' values, will be detected by the MAC check [28]. The authentication of the shared values is postponed to the output phase of the protocol.

SPDZ's implementation details:

- Development language: Python, C++
- Parties supported: 2+ parties
- Adversaries tolerated: semi-honest and malicious adversaries

3.1.3 SCALE-MAMBA

SCALE-MAMBA [29] uses the building blocks of the SPDZ system and improves upon them to implement a maliciously secure two-phase hybrid protocol. However, it resembles a production environment a lot more than SPDZ. The system consists of three main

sub-systems: the offline phase, the online phase, and a compiler. But, unlike SPDZ the offline and online phases are fully integrated, in other words, they are fully combined with one another. This was done so that the system is almost secure out of the box, meaning a certain level of security has already been achieved when the product is first installed and no configurations have been done yet.

It is important to note that to set up a secure channel, users first need to produce a mini certificate authority to be able to run a computation. On another note, the code is written using the framework's special language MAMBA, a Python-like language, and is converted into a byte-code by the compiler, which is then executed by the online/offline phase. SCALE-MAMBA works by offloading all public-key operations to the offline pre-processing phase. Afterwards, it generates three types of shared randomness to use during the execution of the hybrid protocol, which has been previously explained. A major change from SPDZ is the possibility of defining a specific way we want the system to handle the input and output [14].

The framework's flexibility and strong security guarantees often make up for its significant computing requirements, making it a common MPC recommendation. For example, if a malicious adversary is detected by the tool, the computation is aborted immediately.

SCALE-MAMBA's implementation details:

- Development language: Python, C++
- Parties supported: 2+ parties
- Adversaries tolerated: semi-honest and malicious adversaries

3.2 ML-Specific MPC

On the other hand, ML-specific protocols are a kind of special-purpose protocols, which are highly optimized when compared to general-purpose frameworks. ML-specific MPC can train ML models using private data held by different parties. It can also perform inference on encrypted data to analyze and study it. These elaborate tasks are all done while preserving the privacy of the data and its owners, which can lead to a very costly protocol execution. From the least complex to the most complex, we provide some succinct information about SecureNN, FLASH, and TF-Encrypted.

3.2.1 SecureNN

SecureNN [30] is a three-party hybrid protocol that utilizes an additive secret sharing scheme and was made specifically for neural networks. It is capable of training both Fully-Connected and Convolutional Neural Networks (FCNN and CNN) with an accuracy higher than 99%, being the first to do so for CNNs. As previously mentioned, a hybrid protocol provides supporting sub-protocols for common operations. Well, SecureNN provides matrix multiplication, select share, private compare, share convert, and compute MSB (Most Significant Bit). The framework can also implement different neural network training functions, such as ReLU, derivative of ReLU, division, Maxpool, and derivative of Maxpool.

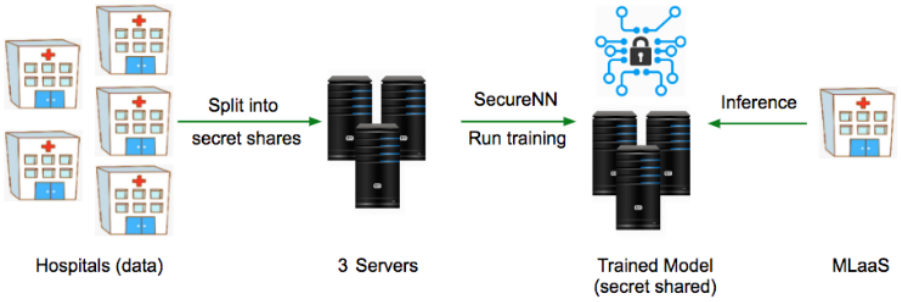


FIGURE 3.3: Architecture of the SecureNN protocol from [30]

To better understand the inner workings of this tool, we will be taking a look at an example given in its respective paper [30] and is illustrated in figure 3.3. Let us suppose there are M data owners that wish to execute training over their joint data with the help of three computation servers. For starters, the M data owners need to secret share their input data with the servers. Then, with the data now at their disposal, the servers can execute the MPC protocol and interact with each other to train a neural network model. To guarantee its privacy, the servers will maintain secret shares of the model between each other. Even in this format, the NN can still be used to perform inference on any new encrypted input. This method keeps the model, the input, and the output secret from other parties, as well as the servers.

SecureNN was the first framework to achieve a faster and more efficient solution than previous two and three-server protocols. This was due to the removal of unnecessary garbled circuits that lead to an improvement in communication complexity, especially when non-linear functions like ReLU are in use. Additionally, SecureNN guarantees the

security notions of privacy and correctness against a semi-honest corruption of a single server or a set of clients, as well as the notion of privacy against a malicious server.

SecureNN's implementation details:

- Development language: C, C++
- Parties supported: 3+ parties
- Adversaries tolerated: semi-honest and malicious adversaries

3.2.2 FLASH

FLASH [31] was one of the first privacy-preserving machine learning frameworks to achieve the strongest security notion of guaranteed output delivery. This four-party protocol can be applied to perform secure server-aided predictions using some ML algorithms, like linear regression, logistic regression, Fully-Connected and Binarized Neural Networks (FCNN and BNN). A BNN is similar to a regular neural network, except the weights used can only have one of two values: 1 or -1. Due to this simplification, a BNN tends to save a lot of memory and time resources when compared to others, but it also has a drop in accuracy. However, unlike SecureNN, it is not prepared to handle CNNs.

Just like previous MPC protocols, FLASH starts with the input sharing process, then advances to circuit evaluation, and ends with the computation of the output. But, it adds a new unique primitive called bi-convey primitive. This method essentially serves two purposes: it enables two parties, A and B, to share a value x with a specific party R and continue with the computation, or in the case malicious intent is identified, it also allows R to identify the corrupt party among A and B. Before FLASH, most of MPC's real-time applications involved a small number of parties and only ensured the security notions of fairness and guaranteed output delivery in the honest majority setting of the protocol. However, with a four-party framework, there is no need for a broadcast channel to perform the computation. Instead, when a message requires an agreement, a simple honest majority rule over the other three parties is enough. Furthermore, expensive public-key primitives can be avoided by implementing a new secret sharing scheme known as mirrored-sharing. This scheme allows two disjoint sets of participants to run the computation and perform an effective validation in a single execution. Overall, FLASH improved communication efficiency extensively.

FLASH's implementation details:

- Development language: C++
- Parties supported: 4 parties
- Adversaries tolerated: semi-honest and malicious adversaries

3.2.3 TF-Encrypted

TF-Encrypted [32] provides encrypted deep learning. It is an open-source library built on top of TensorFlow, which allows for non-experts to implement privacy-preserving ML as quickly as possible, without needing extensive knowledge about complex cryptographic or ML operations, or even distributed systems. Under the hood, it integrates secure multiparty computation alongside homomorphic encryption. The main objectives of this framework are the training of encrypted neural networks using encrypted data and the inference of encrypted inputs.

For the first case, let us suppose a hospital wants a skin cancer detection system to improve its services, but it cannot share any private patient data with data scientists [35]. To get around this problem, TF-Encrypted allows the hospital to share an encrypted format of the data. This way, the data scientists can train, predict, and validate a skin cancer detection model for the medical facility, while maintaining the privacy of the patients. Figure 3.4 displays an illustration of this type of encrypted training.

FIGURE 3.4: Encrypted training performed by TF-Encrypted from [35]

For the second case, let us suppose a new patient sends a picture of his skin lesions in an encrypted format to the TF-Encrypted model [35]. The model will then perform a private prediction without decrypting the image during computation. Figure 3.5 shows a simplified view of encrypted predictions with public training.

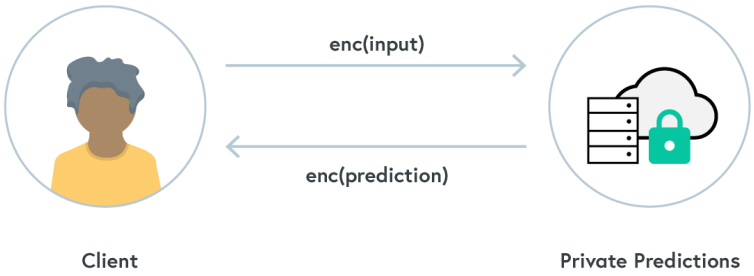


FIGURE 3.5: Encrypted predictions with public training performed by TF-Encrypted from [35]

Nowadays, TensorFlow is one of the most popular libraries for constructing ML models that are then used to perform local or distributed computations. It is mainly used with Python and it offers two main features that explain why it is such a popular solution: an optimized engine and a high-level interface [32]. TensorFlow's engine implements optimizations like lazy evaluation, where the evaluation of an expression is delayed until its value is needed, and multi-core processing, which increases performance by enabling the processing of multiple tasks concurrently.

On the other hand, the high-level interface hides lower-level operations like networking from the programmer. This is extremely beneficial to TF-Encrypted since socket networking operations are an essential component of MPC protocols and are very easy to get wrong. The interface is also capable of assigning operations and tensors that make up a computation to different machines/computing servers. Tensors are essentially multidimensional arrays that are used to calculate arithmetic operations. Furthermore, tensors held by one server may be used by another since it is the responsibility of the TensorFlow runtime to insert the necessary send/recv operations [36]. The runtime also has the ability of choosing a communication technology optimized for the current execution environment.

Since TF-Encrypted is built on top of TensorFlow, it gains all of its advantages. However, only by implementing the former do we obtain a secure computation. For those used to programming with the TensorFlow and Keras libraries, it is extremely easy to switch to TF-Encrypted since the neural networks are built using a similar language to Keras. Nevertheless, it is important to note that TF-Encrypted is not yet compatible with TensorFlow 2.0, so older versions must be used when programming the neural network. On the

same note, we also need to make use of Python 3.6. In addition, the tool takes advantage of a secure random operation that provides a cryptographically strong random number generator for improved security.

Considering that TF-Encrypted performs training in an encrypted format, there is a significant computational overhead when compared to other standard training methods. But, it is a disadvantage that is acceptable when the security of the model and data is more important in a given context. Another drawback of TF-Encrypted is the inability to use TensorFlow functions only present in newer versions. Additionally, the framework commonly applies approximations inside the neural networks to improve speed at the cost of accuracy. However, an advantage of TF-Encrypted is that, just like SCALE-MAMBA, if it detects a malicious adversary during the computation, it will abort its execution immediately. The architecture of TF-Encrypted can be seen in figure 3.6.

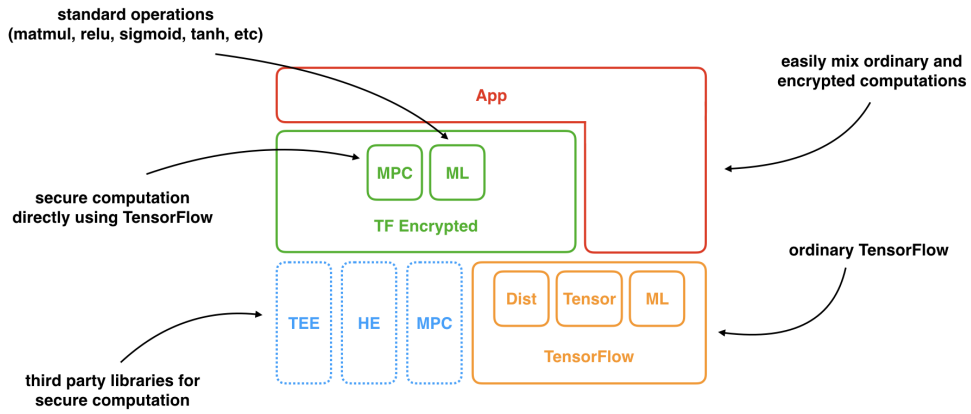


FIGURE 3.6: Architecture of TF-Encrypted from [37]

This framework is integrated with three protocols to perform a secure computation, in a local or distributed environment. These three protocols are ABY3, Pond, and SecureNN. So, to better understand the inner workings of TF-Encrypted, we refer to the SecureNN example given previously.

TF-Encrypted's implementation details:

- Development language: Python
- Parties supported: 2+ parties
- Adversaries tolerated: semi-honest and malicious adversaries

3.3 Other Security-Based Approaches

As a PPML technique, MPC is a type of secure computation that is incredibly powerful and can be applied to a wide range of areas and industries. For example, current financial institutions need to perform routine analysis of large amounts of data stored across different servers or devices, such as credit card fraud detection. However, due to the sensitive nature of the financial data, it can not be stored in a single database and risk a data breach. To solve this problem, Byrd and Polychroniadou [2], proposed the application of federated learning. This MPC protocol enables a set of clients, working with a trusted server, to collaboratively learn a shared machine learning model while keeping each client's data within its own local systems. With this approach, the risk of exposing sensitive data is much smaller.

On the other hand, MPC protocols can also be useful for healthcare purposes. The study done by Şahinbaş and Catak [1] notes that the increasing use of Internet-of-Things technology in healthcare improves the monitoring of chronic diseases, early diagnosis and treatment, rapid intervention, among others. However, the data shared in the digital environment to perform the previous tasks also faces the risk of privacy leakage. So, they proposed a MPC model based on federated learning to overcome the privacy issues, while maintaining a high performance on data analysis.

Considering the widespread use of social networks nowadays, the study of these virtual infrastructures is essential for the field of network science and analytics. However, acquiring the necessary data is extremely difficult due to the associated privacy concerns. Users do not want to share their sensitive information when there is a risk of data leakage. To address these complications, Kukkala, Saini and Iyengar [3], proposed a MPC protocol that can be used to study the behavioral aspects of individuals while guaranteeing the privacy of their sensitive data. Before releasing the sensitive data to the public, the protocol performs naive anonymization on a distributed network without the use of a trusted third party.

Chapter 4

Application Scenarios

This next chapter presents two scenarios where ML-specific MPC was applied to analyze network traffic in a privacy-preserving way. The application scenarios mentioned here are named Torpedo and C2 Traffic Detection. Their resulting systems are presented, including their main goals, design details, as well as dataset contents and neural network architectures. Afterwards, similarities between the two are enumerated to more easily find optimization techniques that can be applied to both scenarios.

4.1 Torpedo

Nowadays, Tor [38] is considered the most popular anonymous network in the world since its release in 2002. It is free and available to everyone that wishes to defend their online privacy or gain censorship-free access to information. Tor implements advanced security and obfuscation techniques to block third-party trackers, prevent surveillance and fingerprinting, which is provided by multi-layer encryption that hides TCP traffic. These traits are the ones that made Tor popular, especially among journalists, whistleblowers, and political dissidents.

Tor can also be used for deploying Onion Services (OS), a kind of infrastructure that provides anonymous, censorship-resistant online services, like, for example, whistleblowing websites and news outlets. OSes hide the IP addresses of both itself and the user currently accessing its respective online service, to preserve their anonymity. Figure 4.1 illustrates how an OS session is established between a client and an OS.

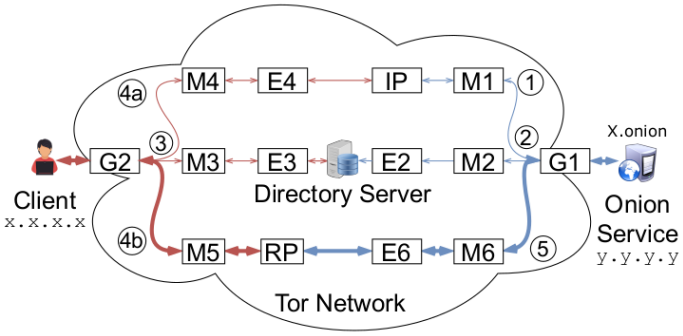


FIGURE 4.1: Onion service session established between a client and an OS from [5]

To set up an onion service session, the following steps are essential [39]:

1. The OS will start by contacting a series of Tor nodes and recruit them as introduction points by establishing anonymized long-term circuits to them;

2. Afterwards, the OS will construct a descriptor containing a list of its introduction points. The descriptor will be published in a distributed hash table (directory) belonging to the Tor network;

3. Then, the client will look up the onion address of the respective OS he wishes to use, and find the necessary introduction point identifier;

4. The client chooses a Tor relay to act as the rendezvous point and establishes a circuit with it. The rendezvous point will relay encrypted messages between the client and the OS, and vice-versa;

5. Finally, the OS will establish a circuit with the rendezvous point chosen by the client.

However, even though this entire process is considered secure, there is a lack of research aimed at understanding if Tor's services are actually safe from deanonymization attacks, which would mean the end of Tor anonymity. For the purpose of studying this possibility, Torpedo was developed to fully deanonymize OS user sessions through a traffic correlation attack. Torpedo is a distributed system that can be deployed by a small group of colluding Internet Service Providers (ISP) in a federated fashion [5].

Each ISP will capture multiple flow samples generated during OS sessions. Afterwards, a deanonymization query can be issued. The system will then analyze the traffic collected by the ISPs through the application of privacy-preserving machine learning techniques. In the end, the system will return a list of correlated flow pairs. A flow pair

matches two flows that are likely to represent ingress and egress segments of a Tor circuit. For each flow pair, Torpedo will return information about the sender and receiver IP addresses of each segment, the date and time of the transmission, and a score that indicates the probability that both flows belong to the same Tor circuit. Figure 4.2 shows the architecture of Torpedo's execution pipeline.

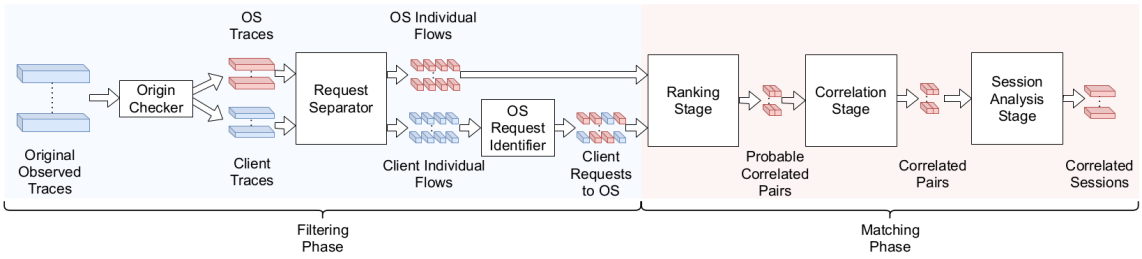


FIGURE 4.2: Architecture of Torpedo's execution pipeline from [5]

The Torpedo pipeline is divided into two phases: the filtering phase and the matching phase. While the filtering phase focuses on pre-processing Tor flows in different network vantage points to generate valid flow pairs, the matching phase will identify sessions with multiple correlated requests involving the same client and OS.

The filtering phase contains three stages:

- **Origin checker** distinguishes flows generated by regular Tor clients from flows generated by OSes, so that the tool can focus on flow pairs that involve a client and an OS. This is possible due to differences in connection patterns between Tor clients and OSes. More specifically, OSes generate a lot more network traffic since they have to publish their onion address in Tor directories and establish multiple circuits to the different introduction points. Finally, to identify the origin of each flow, statistical features such as packet lengths and packet inter-arrival times were studied with the help of an ML-based classifier.

- **Request separator** isolates individual page requests from clients and responses from OSes. This is done in a way that allows all bursts corresponding to a single request to be connected and assembled into a single sample, which is used in the following stages.

- **OS request identifier** is responsible for identifying all requests issued by Tor clients towards regular websites, meaning non-OSes. Considering that connection to an OS implies a larger volume of network traffic and significant differences in burst

volumes, this stage uses an ML-based classifier to easily detect whether a client will connect to an OS or a regular website. This last kind of access is excluded from further analysis.

At the end of the filtering phase, Torpedo receives a set of Tor flows sent from specific clients and going towards OSes, and vice-versa. Meanwhile, the matching phase is also composed of three stages:

- **Ranking stage** scans the flow pairs obtained at the end of the filtering phase to narrow down the search space of potentially correlated requests. After analysis, each flow pair receives a score ranging from 0 to 1. This score represents the confidence of Torpedo that the pair is actually correlated. Only pairs with a score higher than the established pass-through threshold can be selected as candidates.
- **Correlation stage** processes the previously selected candidates using a convolutional neural network that is capable of predicting if a flow pair is correlated or not. Each pair is then assigned a new score also ranging from 0 to 1. In this case, the score represents the probability of a client-generated flow to be correlated with a specific OS-generated flow.
- **Session analysis stage** analyses the flow pairs obtained during the correlation stage and decides whether a set of flow pairs is enough to identify a browsing session between a client and an OS.

For this project, we analyzed and optimized the convolutional neural network used during the correlation stage, since it was already set up to run with MPC protocols.

4.1.1 Dataset

Torpedo uses a dataset that accurately represents Tor OSes interactions. In a laboratory environment, a total of 35 Virtual Machines (VM) were set up. Fifteen VMs were used to host OSes, which ran on an isolated Docker container, and the remaining twenty VMs acted as Tor clients. With this environment, it was possible to emulate a set of concurrent browsing sessions towards OSes and regular webpages via Tor. Even though there are some datasets that capture clients interacting with regular webpages through Tor, there is a lack of datasets that accurately represent clients accessing and utilizing OSes. Torpedo

actually tried to mitigate this lack of research and focused on collecting Tor OS interactions.

But, before generating the dataset, some information had to be considered. Firstly, the distribution of requests made to OSes is highly skewed towards a small number of popular OSes. So, some sets of OS interactions, generated for Torpedo, actually had to mirror realistic popularity rates. Secondly, there are two distinct features of real OS webpages that allow its full reconstruction: total page size and the number of extra requests. These extra requests include assets such as JavaScript, or CSS files and images. For Torpedo, fifteen webpages were created according to the distribution of the previous two features, so that they can better resemble actual pages.

During the emulation of the necessary browsing sessions, the VMs acting as Tor clients repeatedly accessed the OSes hosted in the other VMs and the top 50 Alexa pages. This was done in parallel during an one hour time-span, with approximately fifteen sessions per minute. Once a client's session ended, they waited a total of 60 ± 5 seconds before starting a new one. Following the popularity rates, each VM acting as a Tor client had a 75% chance of issuing a new session towards an OS and a 25% chance of placing a session towards one of the randomly selected top 50 Alexa websites.

In the end, a balanced dataset consisting of traffic samples corresponding to the launch and startup of 1000 Tor clients and 1000 OSes was obtained. The training set used during this project has 19539 traffic samples, while the testing set has 13026 samples. The dataset contains eight numerical traffic features extracted from a pair of flows. These features include: incoming packets' inter-arrival times, outgoing packets' inter-arrival times, incoming packets' sizes, and outgoing packets' sizes for each traffic flow. All of the previous features apply to clients and hosts.

As a final note, this project followed the Tor research safety board recommendations [40] to avoid ethical problems related to private user data collection. The traffic correlated by Torpedo only includes browsing sessions generated for the very purpose of experimentation and does not jeopardize the safety and privacy of normal everyday users.

4.1.2 Neural Network Architecture

Figure 4.3 displays the Convolutional Neural Network (CNN) used during Torpedo's correlation stage. The NN is composed of two convolutional layers and three fully-connected layers. For starters, the input layer has a size of 2400 neurons. This corresponds to the

number of features extracted from a flow pair (8) multiplied by the number of packets the model is trained on (300). The first convolutional layer uses 1000 neurons, while the second convolutional layer utilizes 500 neurons. As for the three dense layers, they have a size of 1500, 400, and 50 neurons, respectively. The output layer contains only one neuron. For all dense layers, the ReLU activation function is implemented, except for the output layer, where the Sigmoid cross entropy loss function is applied. After each convolutional layer, a Maxpool operation is used to downsample the input along its spatial dimensions (height and width). On the other hand, after each dense layer, a dropout operation with a rate of 40% is executed. During training, the Adam optimizer with a learning rate of 0.01% is used.

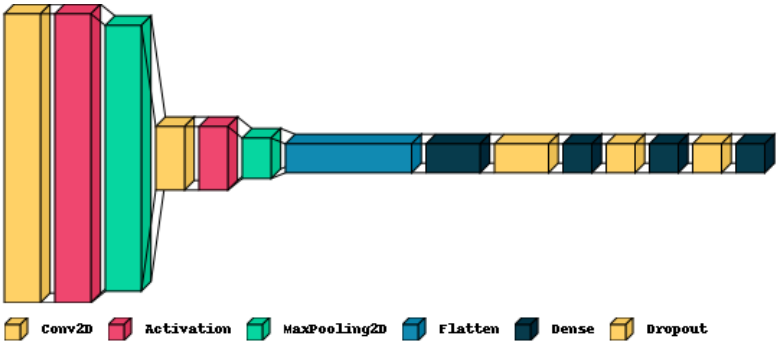


FIGURE 4.3: Torpedo neural network architecture

4.2 C2 Traffic Detector

Currently, malicious Command and Control (C2) servers communicate with infected machines to provide new attack orders or to receive stolen data obtained by said machines. However, to avoid detection by payload inspection, most C2 traffic resorts to encrypted communications, such as TLS. So, in order to detect this type of malicious traffic, it is common to use server-side certificate blacklisting. There already are many collections that enumerate malicious certificates, like the SSL Blacklist project, which lists SSL/TLS certificates employed by botnet C2 servers.

Unfortunately, the previous tactic is not enough. Attackers can resort to free TLS certificate providers and easily use new certificates for their C2 servers. Nowadays, detectors employing deep learning techniques to analyze traffic and its features are being used to

strategically circumvent this problem. Being able to detect malicious C2 traffic is invaluable for when attackers change already black-listed items, such as IP addresses and server certificates.

However, these deep learning algorithms are especially vulnerable to adversarial attacks. An adversarial attack is essentially a method for generating adversarial examples, a kind of input specifically created to cause a ML model to make a mistake in its predictions. The catch is that, to the human eye, adversarial examples resemble valid inputs. So, in this case, attackers try to modify the behavior of C2 traffic between an infected machine and the C2 server to make detection harder for the model, as well as administrators. Figure 4.4 shows an illustration of an adversarial example, where a ML classifier confused the image of a panda with a gibbon.

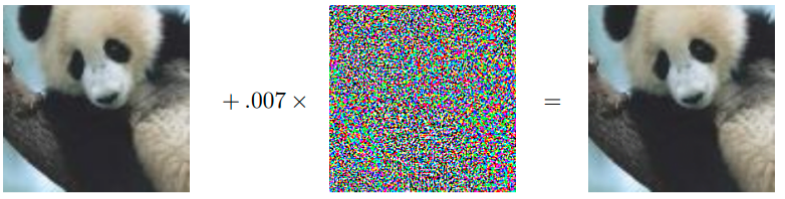


FIGURE 4.4: Illustration of an adversarial example from [41]

Even though the previous attack tactic is extremely efficient, it is also extremely expensive to develop. Modifying the behavior of traffic to act in a very specific way requires a lot of time and resources, which are often not available to most attackers. A more frugal and more attractive solution is to simply alter some or all traffic features by overwriting payloads.

Considering the previous restrictions, this work tries to level the playing field between an attacker and a defender. Figure 4.5 illustrates such an attack and defense architecture using a C2 traffic detector. The adversarial proxies monitor the traffic exchanged between the server and the original malware, and when necessary, modify traffic features in the hopes of fooling the detector. Some possible alterations might involve delaying or adding packets at the end of a TCP connection. On the other hand, the C2 detector is set up on one end of the Intrusion Detection System (IDS). While the IDS uses rule-based modules to identify blacklisted certificates, the C2 traffic-based detector is responsible for detecting not-yet-blacklisted C2 traffic.

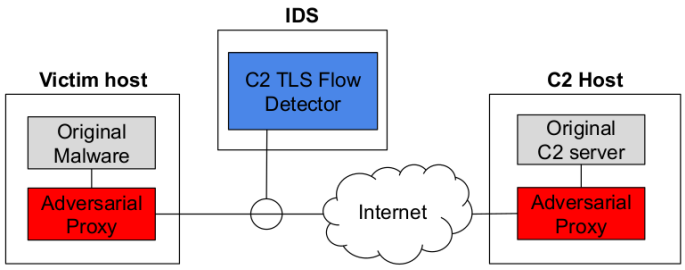


FIGURE 4.5: Attack and defense architecture using the C2 traffic detector from [6]

With this system, Morla and Novo [6][7] exemplify how using different sets of features for attacking, and others for defending and hardening the ML model affects the detection of malicious C2 traffic. The process of hardening the model consists of increasing its robustness against adversarial examples by training it with similar examples. A white-box adversarial learning method was used throughout the project. This means that the attacker has access to the model's parameters.

It is important to note that they are assuming that the attackers cannot change the original malware code. Instead, the adversaries will either: apply a proxy or add additional code. The proxy-like approach, will lead to changes in the traffic features, such as increasing the duration of flows. On the other hand, adding new code at the end of the original malware code, will lead to the creation of additional TLS records, which will be ignored by the C2 server, but will result in an increase in bytes' and packets' count, as well as an increase in the duration of the flow.

4.2.1 Dataset

The C2 traffic detector uses a public dataset provided by the Malware Traffic Analysis (MTA) project [42]. This website contains a wide range of detailed content related to common malware families, which is extremely useful to provide insights on network-based detection. Around 508 .pcap files consisting of captured network traffic associated with malicious C2 communications were extracted from the MTA website. All these files contained 20747 TLS samples, including C2 and non-C2 traffic.

Then, to detect C2 traffic within the previous collection, the Suricata IDS was configured with rules to identify malicious TLS traffic flows. The rules were created with the help of the SSL Blacklist project, which contains a list of SSL/TLS certificates employed by malware C2 servers. The IDS detected 7672 malicious TLS flows, while the remaining

13075 TLS flows were considered non-malicious traffic. The malicious flows together with half of the non-malicious traffic were used to create a 50%-50% balanced dataset.

Core Set	*Features 3-14, 17-28, 31-37*
3-6 ; 17-20	Total, RST, ACK, pure ACK packet counts
7 ; 21	Unique bytes
8-9 ; 22-23	Data segment and byte counts
10-11 ; 24-25	Retransmitted data segment and byte counts
12 ; 26	Out of sequence segment counts
13-14 ; 27-28	SYN and FIN packet counts
31	Flow Duration
32 ; 33	Rel. time of first payload
34 ; 35	Rel. time of last payload
36 ; 37	Relative time of first ACK

FIGURE 4.6: C2 traffic detector's core set of features from [6]

The dataset contains 86 numerical features extracted from traffic flows by Tstat, a TCP statistics and analysis tool [43]. While incomplete flows were ignored, as defined by Tstat, non-sampled and sampled flows were normalized. These features include: pure acknowledgement packets' count, flow duration, TCP options, data segments' count, bytes' count and others. Figure 4.6 enumerates some of the most noteworthy features.

4.2.2 Machine Learning Architecture

Figure 4.7 displays the Fully-Connected Neural Network (FCNN) used by the C2 traffic detector. The NN is composed of three fully-connected layers. The input layer has a size of 86 neurons, which corresponds to the number of features in the dataset. Meanwhile, the three hidden layers have 2048, 1024, and 512 neurons, respectively. The output layer has a size of two neurons. Each layer implements the ReLU activation function, except for the output layer that uses a Softmax activation function. After each dense layer, a dropout operation with a rate of 20% is applied. During training, the Adam optimizer with categorical cross entropy loss function is used.

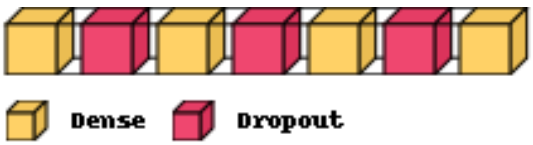


FIGURE 4.7: C2 traffic detector neural network architecture

4.3 Common Pipeline

Torpedo and the C2 traffic detector are both used to analyze network traffic in a privacy-preserving manner. Despite classifying different types of traffic flows, there are some similarities to be found between the two. These similarities result from the resemblance between inputs, which in both cases are data flows. This reinforces our argument that there may be optimizations common to both scenarios, as well as other problems related to traffic flow analysis.

For starters, both scenarios contain datasets with numerical features related to packet transmission times and packet sizes. Despite the C2 traffic detector's dataset having a lot more features than Torpedo's dataset, does not mean that there are not some resemblances between the two. As previously mentioned, the Torpedo dataset contains features related to client's and host's incoming packets' inter-arrival times, outgoing packets' inter-arrival times, incoming packets' sizes, and outgoing packets' sizes. The C2 dataset also has some features with similar information like flow duration, packets' count, data segments' count, bytes' count, and more. When studying the sizing and timing data of Torpedo, we have to study all of the corresponding features within the C2 dataset.

Secondly, all features are of the float64 type. A float64 is a double precision number stored in 64 bits. This floating-point format allows us to reach a higher resolution, but it also means that we have to process a larger volume of data and require more extensive memory/CPU resources. Overall, the training and prediction stages of both the Torpedo's and the C2 traffic detector's models are much more costly than other models that might use float32 or even float16 formats. This will be further explored in the next chapter.

Additionally, both scenarios implement the ReLU activation function. An activation function decides whether an artificial neuron should be activated or not. Basically, it makes the decision of whether the node's input is important or not to the output of a neural network. Activation functions are essential to introduce non-linearity into a NN, meaning that without it all layers would behave the same way. Usually, the hidden layers apply the same function, while the output layer implements a different one. Nowadays, the ReLU function is the most popular activation function. It is used in most FCNNs and CNNs, since it achieves better performance and makes the training process much easier. The function returns a value if it is positive, or it returns 0 if the value is equal to 0 or less. Since Torpedo and the C2 traffic detector use a CNN and a FCNN, respectively, it

would explain why both resort to a function with the highest success rate. Since ReLU is the most popular choice, there is no need to study possible alterations.

Finally, Torpedo and the C2 traffic detector apply the Adam optimizer. Choosing the correct optimizer for a deep learning model is extremely important since they can improve the training speed and performance of the model. They achieve this by changing the model's weights or learning rate to minimize loss as much as possible. As expected, there are many different kinds of optimizers, but Adam is one of the most popular. It is a gradient descent algorithm that essentially tries to find individual learning rates for each weight parameter. It requires less memory resources than others, but is also very fast, flexible, and robust. The Adam optimizer is a good choice for dealing with a lot of data and parameters. This explains the decision to apply it in both Torpedo and in the C2 traffic detector. Just like ReLU, there is no need to study possible alterations to the optimizer since the best choice has already been made.

Chapter 5

Testbed Implementation

Our implementation scenario assumes that the data has already been gathered and that the ML model has already been trained. We off-loaded the inference process to an unreliable third party, to then obtain the performance values of said process. We did this by applying a MPC framework known as TF-Encrypted. First, the model is trained during the setup stage, in an insecure manner. Then, the model is secret shared between players acting as computation servers. Afterwards, the model receives our inference queries, calculates the output and adds all the shares to convert it to a human-readable format.

We want to obtain the performance baseline values of the previous system to then check how the MPC protocol reacts to different changes and optimizations. These changes are all applied to the data, while the ML model remains unchanged. Therefore, we built a distributed testbed to achieve realistic results as much as possible. A distributed system allows us to more easily analyze the performance of MPC protocols since their bottleneck is due to high latency, especially when it comes to more practical applications. The architecture of the distributed system can be seen in figure 5.1.

In this chapter, we will provide an overview of how the installation and configuration of TF-Encrypted was done. The installation of the MPC tool was performed in a way that allows the two previous application scenarios to be run in different execution environments, which include a local plaintext execution, a secure MPC execution where all participants run in the same local machine, and a distributed MPC execution.

Then, we will show the baseline values obtained for both the Torpedo and the C2 traffic detector ML model in each environment. The baseline values consist of the inference time, measured in seconds, the recall, precision, and F1-score classification metrics. With

the baseline established, we can compare the performance results of the neural networks before and after applying some optimizations, which will be enumerated below.

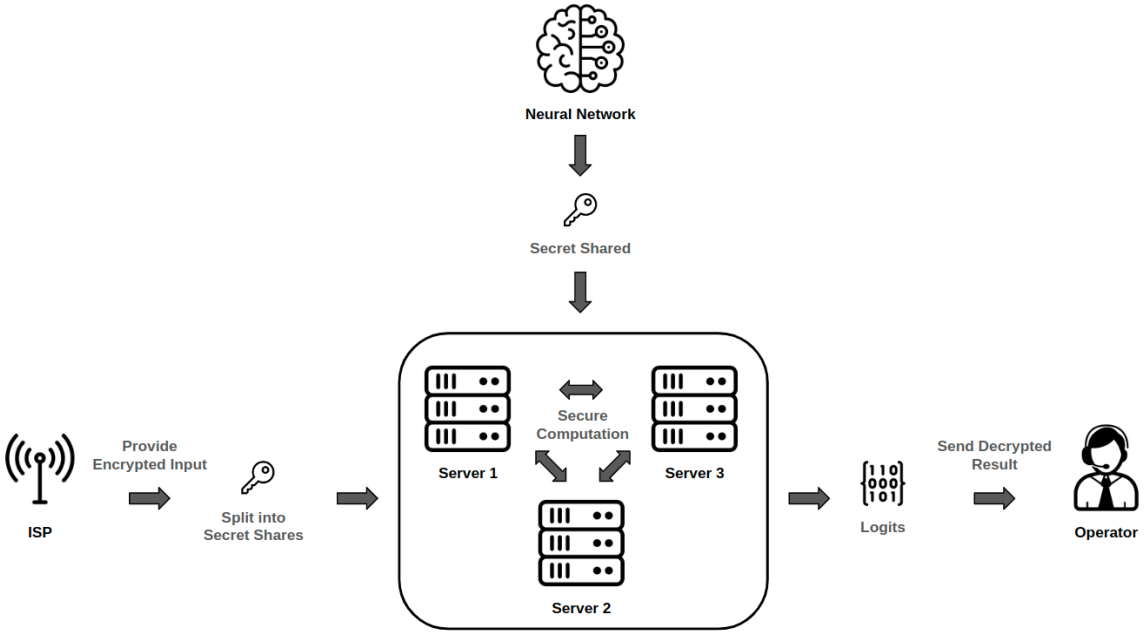


FIGURE 5.1: Distributed system architecture

5.1 TF-Encrypted

Firstly, TF-Encrypted [44] was installed on a local computer. This ML-specific MPC tool was selected due to its accessibility and compatibility with TensorFlow and Keras, two libraries that were used to build the ML models for both Torpedo and the C2 traffic detector. However, as mentioned in the state of the art chapter, TF-Encrypted is only compatible with Python version 3.6 and TensorFlow version 1.15. All the other necessary packages are enumerated in the requirements text file that comes with the MPC framework. For convenience sake, a virtual environment was created with the required versions of Python, as well as other packages, to execute the tool.

With the framework finally configured, we needed to understand how to execute the Torpedo's and the C2 detector's neural networks with TF-Encrypted. These NNs were already built and configured inside Python scripts, which implemented the TensorFlow API. Therefore, we only had to update the code provided, so that the prediction process could run using a MPC protocol. But, to implement such privacy-preserving ML techniques within TF-Encrypted, a certain code structure needs to be followed. A simplified snippet of this code is shown in listing 5.1.

```
import tensorflow as tf
import tf_encrypted as tfe

class ClientISP:
    def provide_input():
        return prediction_input

class ClientLEA:
    def receive_output(logits):
        return tf.print(tf.sigmoid(logits))

def load_weights():
    return model.get_weights()

isp = ClientISP()
lea = ClientLEA()
weights = load_weights()

with tfe.protocol.SecureNN():
    prediction_input = isp.provide_input()
    model = """ Build Keras model """
    logits = model(prediction_input)

prediction_op = lea.receive_output(logits)

with tfe.Session() as sess:
    model.set_weights(weights, sess)
    sess.run(prediction_op)
```

LISTING 5.1: Example of a MPC protocol execution in TF-Encrypted

According to the code structure shown previously, the execution of a MPC protocol, using TF-Encrypted, includes the following steps:

1. Firstly, TF-Encrypted creates an ISP client and a LEA client;
    - The Internet Service Provider (ISP) client takes on the role of input provider, while the Law Enforcement Agency (LEA) client acts as the result receiver. It is important to note that the names given to the clients derive from the Torpedo application scenario. However, this does not mean that we only consider scenarios involving ISPs and LEAs, they are simply a means to exemplify real world parties that might make use of MPC protocols.
2. Then, TF-Encrypted loads all layer weights of the ML model;

- The ML model has already been trained, so the weights are loaded from the disk, where they have been previously saved.

3. Afterwards, it establishes the MPC protocol to use during prediction;

   - As mentioned in the state of art, TF-Encrypted has three protocols at its disposal: ABY3, Pond, and SecureNN. For all computations, we used the SecureNN protocol since it is the one we have studied and know best. Therefore, to better understand how TF-Encrypted is performing secure computations, we only need to refer once more to the SecureNN execution example given in chapter 3.

4. With the MPC protocol established, the ISP client provides its inputs;

   - To do this, it starts by creating a data pipeline to iterate over the inputs. It then loads and pre-processes the encrypted input data. After being processed, the data is sent in secret shares to parties acting as the computation servers. The ISP also prints the expected results of the prediction operation on the terminal.

5. Then, the model is built, so that it can perform the necessary predictions;

   - Just like the inputs, the model is kept as secret shares between the computation servers.

6. The model generates the logits;

   - Logits are essentially a vector of raw (non-normalized) predictions that the ML model generates after computation.

7. Finally, a new TF-Encrypted session is created to run the prediction operation.

   - This process involves setting the model's weights, decrypting the calculated logits by turning them into a human-readable result, and sending it to the LEA client's terminal.

For Torpedo, the LEA client will receive a set of scores between 0 and 1. Each score indicates the probability that a given flow pair represents a Tor circuit. In other words, the probability that a flow pair identifies a browsing session between a client and an onion service. Figure 5.2 displays a set of results obtained after computing Torpedo's neural network with TF-Encrypted.

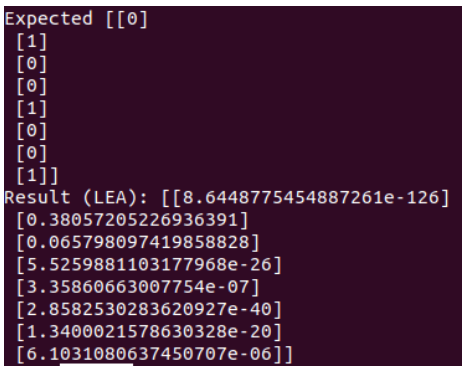


FIGURE 5.2: Example of results obtained after executing Torpedo's neural network with TF-Encrypted

For the C2 traffic detector, the LEA client will receive a set of scores also between 0 and 1, each representing the probability that a specific traffic flow is a malicious C2 traffic flow. Figure 5.3 shows a set of results obtained after computing C2 traffic detector's neural network with TF-Encrypted.

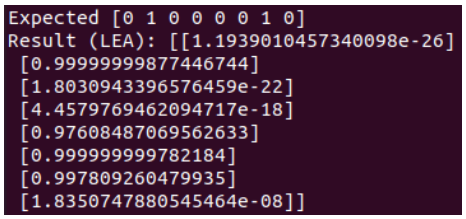


FIGURE 5.3: Example of results obtained after executing the C2 detector's neural network with TF-Encrypted

The difference between the previous piece of code and a normal NN computation resides mainly in the use of the TF-Encrypted Keras library instead of the TensorFlow version. This guarantees that the whole process is encrypted and secure. However, the load_weights function, which loads the model's weights from the disk, must use ordinary TensorFlow to work properly. It is the only operation that runs locally, in plaintext, instead of running in an encrypted format.

The previous code snippet guarantees a local MPC execution of the neural networks, but it is not yet truly secure. It is only simulating a secure computation by running everything on the local machine. In this case, different threads are being created by TF-Encrypted to act as the different participants. New players will automatically be created as needed. So, for example, one thread would perform the role of a computation server, while the second one acts as the ISP client and the third one acts as the LEA client. This execution environment is very useful to test the behavior of the players during the computation. However, in this case, the memory is being shared between all threads, meaning

a secure execution of a MPC protocol cannot be reached since there is no privacy between the different players.

5.2 Distributed System

A truly secure distributed computation requires that all players are separate from each other. To achieve this, five virtual machines provided by FCUP were used. Three of those machines act as the computing servers, while the fourth one acts as the ISP client and the fifth one represents the LEA operator. This setup ensures that the players do not share any memory or other resources. It also makes sure that if a party is compromised, the other parties are not affected. Furthermore, TF-Encrypted alongside TensorFlow version 1.15 and Python version 3.6 were installed and configured in all five VMs.

All VMs provided by FCUP run CentOS Linux 8 with 257GB of RAM and an Intel(R) Xeon(R) Silver 4210 CPU at 2.20GHz. Meanwhile, the computer used for the local computations runs Ubuntu 20 LTS with 15GB of RAM and an 11th Gen Intel(R) Core(TM) i7-1165G7 CPU at 2.80GHz.

One important thing to note is that the VMs are connected via Ethernet, instead of the Internet, which is not the ideal setup to simulate a truly distributed system. This also affects the computations results obtained throughout the project since Ethernet tends to offer less latency and faster connection speeds when compared to the Internet. Even so, the Ethernet connection still shows the weight of privacy-preserving computations, which allows us to correctly compare the performance of plaintext executions with secure ones.

Nevertheless, we must also add a new code segment to ensure that all participants are identified and can connect to each other. Listing 5.2 shows how to configure TF-Encrypted to perform a remote computation.

```
networkConfigPath = """Path to network config file"""

config = tfe.RemoteConfig.load(networkConfigPath)
tfe.set_config(config)
tfe.set_protocol(tfe.protocol.SecureNN())
```

LISTING 5.2: Configuration of a remote MPC protocol execution in TF-Encrypted

During the MPC computation, participants will use the Transmission Control Protocol (TCP) to communicate with each other. The TCP protocol is incredibly advantageous

when it comes to data retransmission, congestion control, error detection, and unique identification. This method of unique identification requires that each machine on the network has been assigned with an unique IP address, allowing it to be identifiable over the network. In TF-Encrypted, this is achieved through a network configuration file. Before launching a remote computation, TF-Encrypted needs to know the location of a simple JSON file with an array of the five VMs' personalized names and their corresponding IP addresses and port number. The port number will range from 4440 to 4444.

Even though TCP does not provide any data encryption functions nor does it protect connections against unauthorized access attacks, these security problems are all mitigated by TF-Encrypted itself. As stated in the background chapter, the two main security notions guaranteed by any MPC protocol are privacy and correctness. Privacy assures that no interested party learns anything beyond what is necessary, meaning an unauthorized participant will learn nothing. Additionally, TF-Encrypted already provides the necessary encryption methods. Correctness, on the other hand, assures that each party receives its correct input, so no attacker can modify the information exchanged. Therefore, TF-Encrypted can use TCP to provide communication between parties without compromising its security. The only downside of implementing TCP is its slow start, an algorithm that helps control the amount of data flowing through to a network. This might help with congestion control, but it may also slow down connections, especially encrypted ones.

Afterwards, the only step left is to start a TensorFlow server in each VM by specifying their personalized name, just like in the JSON file. The location of said file must also be indicated. This can be done using the command shown in listing 5.3.

```
python3.6 -m tf_encrypted.player name --config config.json
```

LISTING 5.3: Configuration of a TensorFlow server

However, there was a minor setback when initiating the secure distributed execution. As can be seen in figure 5.4, to connect the local computer to the VM cluster, we must go through a jump server. This is a technical requirement of the machines provided by FCUP.

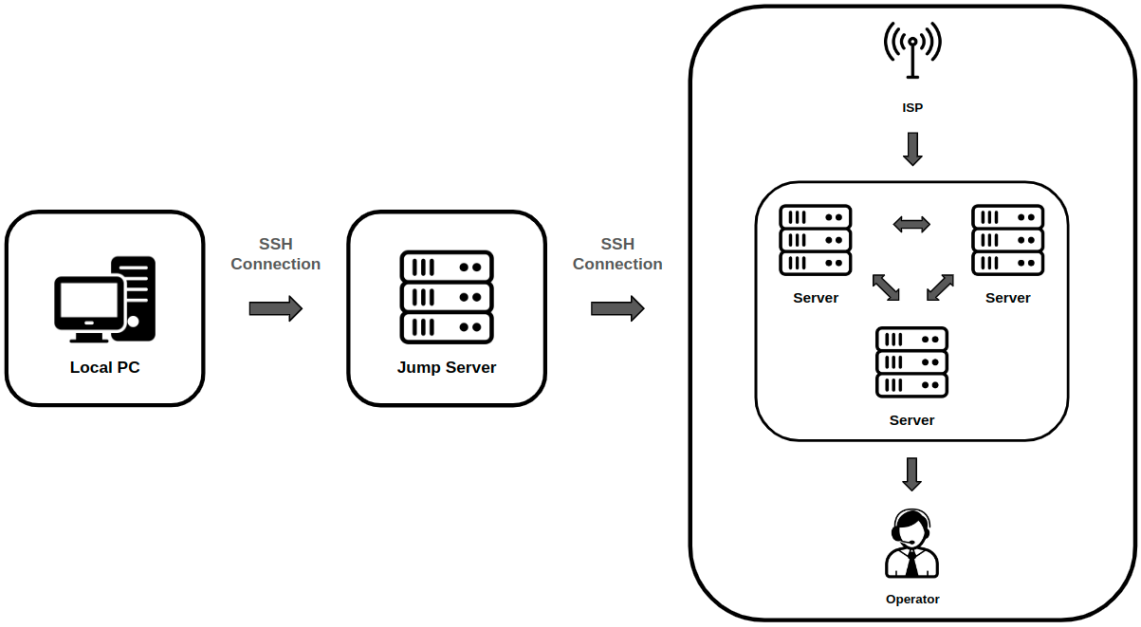


FIGURE 5.4: Connection to distributed system

Due to the jump server, it is not possible to send a command from the local computer to the first VM to initiate the MPC computation. So, when starting a TensorFlow server on the first VM, the Python script will be executed in conjunction with the first command. In other words, it will be the first VM's responsibility to begin the MPC computation. For this to work, we had to copy all Python scripts belonging to Torpedo and the C2 detector to the VM.

5.3 Baseline Values

Now, we have three distinct execution environments: a local plaintext execution, a local MPC execution, and a distributed and secure execution, as well as two application scenarios: Torpedo and the C2 traffic detector.

There are four values necessary to establish a baseline, which allows us to assess the performance of each application scenario in the different environments throughout the project. These baseline values are inference time, recall, precision and F1-score. The inference time is measured in seconds and shows the time necessary for the ML model to perform inference on a test set.

First, we will take a look at the inference time of each application scenario in the three execution environments. These values can be seen in table 5.1.

Scenario	Plaintext (s)	MPC (s)	Distributed (s)
Torpedo	1.44	8.18	44.44
C2	0.63	1.52	4.76

TABLE 5.1: Inference time baseline values per scenario and execution environment

Already we can see a clear difference between the two application scenarios. The C2 traffic detector is much faster at performing inference than the Torpedo system. However, C2's neural network contains only three fully-connected layers, while Torpedo has two more convolutional layers on top of the three dense layers. A larger number of layers, especially convolutional ones, increases the number of weights in the network, which in turn increases the ML model's complexity. Furthermore, CNNs are extremely slow when executing on CPUs since they have to perform expensive matrix multiplications.

Now, we can examine the recall, precision, and F1-score classification metrics for each scenario. These values can be seen in table 5.2.

Scenario	Recall (%)	Precision (%)	F1 (%)
Torpedo	77.9	93.5	84.7
C2	99.9	90.3	94.9

TABLE 5.2: ML classification metrics baseline values per scenario

While the Torpedo NN shows a higher precision than the C2 NN, it also has a lower recall value equal to 77.9 %. Considering that Torpedo is responsible for calculating the probability that two traffic flows belong to the same Tor circuit, the recall must be higher to avoid a larger number of False Negatives. As for the C2 scenario, the precision could be a bit higher to lower the number of False Positives and avoid incorrectly identifying malicious traffic flows. Furthermore, to improve the performance of the privacy-preserving techniques used, we must also try to lower the inference time of each model. With these goals in mind, we will first go over some possible optimization techniques and then check how they affect the overall performance of the Torpedo and C2 neural networks.

5.4 Optimization Techniques

Essentially, neural network optimization revolves around reducing the number of parameters/features used by the model. By utilizing a smaller number of features, the model needs to perform fewer computations, which in turn reduces the inference time. There are different methods to achieve this, but in this work we focused on parameter removal and

parameter quantization. While parameter removal focuses on eliminating unnecessary features that do not contribute to the output, parameter quantization focuses on reducing the memory needed by each feature. Since data is being changed, it is important to train the neural network every time an optimization technique is implemented.

So, in this section, we enumerate some possible optimizations to be applied to ML-specific MPC protocols. The next section shows the results obtained by applying said optimizations and if they were effective or not.

5.4.1 Parameter Removal

The precision of network traffic classification depends on the network features used to train the ML model. A correct selection of features can also lead to a reduced computational load of the model since fewer parameters require less memory resources. Nevertheless, identifying the data that will be the most useful is incredibly difficult, especially when dealing with neural networks. Oftentimes, feature selection for NNs needs to be performed in a manual fashion, which can be quite a lengthy task.

However, some studies related to network traffic analysis have already identified which combination of features would be the most precise in classifying network traffic. These features include the source IP address or port, the destination IP address or port, the time interval between packets, and the size of packets [45].

So, if we try to remove the features unrelated to any of the previous ones, it stands to reason that either the precision of the model improves or the computational load decreases. Unfortunately, this optimization can only be tested correctly using the C2 traffic detector scenario, since Torpedo already has a dataset with only packet features related to times and sizes. But, we will still show the results for Torpedo when removing this kind of data to highlight its impact on the NN's classification metrics.

When performing parameter removal, there are two challenges to face. First, the lengthy task of identifying the useless features that can be removed. And secondly, identifying the limits of the previous procedure. If we remove too many features the accuracy and precision of the ML model can start to decrease more than we intended.

5.4.2 Parameter Quantization

Usually, floating-points are not considered the most important aspect when dealing with neural networks. If the NN model is not performing well, then changing the floating-point format is hardly going to improve it. However, when a certain level of model complexity is reached, the floating-point format can have a significant impact on the model's training times and overall performance.

It has long been established that Fully-Connected and Convolutional Neural Networks (FCNN and CNN) can handle a lower numerical precision with little to no degradation in the classification accuracy [46][47]. Training FCNNs and CNNs requires extensive computational resources, as well as large quantities of labeled data. So, high-precision calculations end up offering no benefit, and have the disadvantage of being slower and less memory-efficient.

There are three main floating-point formats we will be focusing on: FP16, FP32 and FP64. FP16 is known as the half-precision floating-point format and represents a sequence of 16 bits, of which 10 are the precision bits. Likewise, the FP32 is the single-precision floating-point format and represents a sequence of 32 bits, with 23 acting as the precision bits. Finally, FP64 is known as the double-precision floating-point format and represents a sequence of 64 bits, where 53 are the precision bits.

Both the Torpedo and the C2 traffic detector datasets use numerical features of the float64 type. Thus, we can convert the features to float16 or float32, in other words, use the half-precision or single-precision floating-point format, and train the NNs to see how their performance is affected by reducing the memory footprint.

The main downside of this technique is that it usually results in much information loss. Therefore, it is extremely important to choose the correct floating-point format to achieve a balance between information loss and memory requirements.

It is important to note that we only focused on converting the features to other floating-point formats instead of converting them to integers due to the TensorFlow version we had to use. Even though TensorFlow 1.15 has a function to cast data types to integers, it was causing a lot of errors to appear on the neural networks, which prevented their correct training. The cause of these errors was impossible to detect, so we decided to discard this quantization technique.

5.5 Results

For this section, we present and discuss the results obtained after applying each of the previously mentioned optimization techniques. These results were extracted from local plaintext executions of both application scenarios, since a secure version would be too expensive to train over and over again. Furthermore, the training and inference operations were performed on a CPU to mimic the other execution environments that cannot take advantage of a GPU. We consider the inference time of the model, as well as three ML classification metrics: recall, precision and F1-score.

5.5.1 Torpedo

Table 5.3 displays the results obtained for Torpedo after implementing each of the optimization techniques.

Optimization	Inference Time (s)	Recall (%)	Precision (%)	F1 (%)
None	1.44	77.9	93.5	84.7
Rem. times	1.40	65.7	89.7	75.9
Rem. sizes	1.39	72.0	96.6	82.5
FP32	1.41	75.9	96.9	85.1
FP16	1.39	80.3	97.7	88.2

TABLE 5.3: Torpedo's results after optimization

As can be seen in the previous table, by trying to remove the packet inter-arrival times, there is an obvious degradation to the precision and recall metrics, which went down to 65.7% and 89.7%, respectively. On the same note, by eliminating the packet sizes, the precision might improve to 96.6% but it has a negative impact on the recall metric, which is lowered by around 6%. This confirms what we previously stated regarding the importance of the time interval between packets and the size of packets for network traffic analysis [45]. So, for Torpedo there is no advantage in implementing parameter removal optimizations.

Moving on to the parameter quantization techniques, we first tried to apply a single-precision floating-point format, but even though the precision increased to 96.9%, the recall went down 2%. For a network traffic classifier, we do not want a low recall value since that means that the model will produce a lot of False Negatives. Therefore, we can conclude that the FP32 is not the best floating-point format to use.

However, after applying the half-precision floating-point format, we can observe a clear improvement in Torpedo's performance overall, including the inference time. The recall went from 77.9% to 80.3%, while the precision went from 93.5% to 97.7%. This also led to an improved F1-score. It is clear that the lower numerical precision allows the CNN to be more efficient during training.

5.5.2 C2 Traffic Detector

Table 5.4 displays the results obtained for the C2 traffic detector after implementing each of the optimization techniques.

Optimization	Inference Time (s)	Recall (%)	Precision (%)	F1 (%)
None	0.63	99.9	90.3	94.9
Rem. Tstat feat.	0.55	99.9	87.7	93.5
Rem. IP packets	0.54	99.9	87.0	93.0
Rem. TLS rec.	0.54	99.9	91.0	95.3
FP32	0.56	100	89.8	94.6
FP16	0.59	99.9	90.0	94.8

TABLE 5.4: C2 traffic detector's results after optimization

Considering that the C2 dataset contains a lot more features than Torpedo's, we can try more combinations of parameter removal. Firstly, we removed the Tstat features, which include 35 parameters related to client's and server's total packets' count, ACK packets' count, bytes' count, flow duration, retransmitted bytes, among others. Since this set of features contains timing and sizing data, after eliminating it there was a degradation to the precision and F1-score. Both went down to 87.7% and 93.5%, respectively.

Next, we tried to remove every feature related to IP packets, including number of bytes and inter-packet-times. After eliminating 40 parameters, we obtained a lowered precision of 87% and a worse F1-score of 93%. Thus, it is best to keep the IP packets' features because they clearly contribute to the NN's output.

Afterwards, we tried to remove all TLS records data, which also contained the number of bytes and inter-packet-times. After removing 48 features out of the 124 features, we can see that there was no negative impact on the ML classification metrics. While the recall stayed the same, both the precision and F1-score improved slightly to 91% and 95.3%, respectively. Additionally, the inference time also had a small reduction of around 0.10 seconds. This means that the removed TLS features are not that relevant to the NN and can be safely ignored.

Finally, we can take a look at the parameter quantization optimizations. Unlike Torpedo, the C2 traffic detector's ML classification metrics are not positively affected after applying the single-precision and half-precision floating-point formats. For the FP32, the recall is capable of reaching 100% at the cost of a lowered precision of 89.8%. On the other hand, FP16 does not show a great change in the classification metrics. The only upside of both is a small improvement in the inference time. As previously stated, the floating-point format only has a significant impact if the model's complexity has reached a certain level. While the C2 application scenario has a FCNN with only three fully-connected layers, Torpedo is much more complex due to the fact that it uses a CNN. Considering these aspects, we can understand the difference in results.

5.5.3 Final Results

By selecting the best optimizations, we can check if they have any impact on the model's performance during a secure computation. Table 5.5 and table 5.6 display the inference time needed in a secure and distributed environment after optimizing the NNs.

Optimization	Inference Time (s)	Recall (%)	Precision (%)	F1 (%)
None	44.44	77.9	93.5	84.7
FP16	40.60	80.3	97.7	88.2

TABLE 5.5: Torpedo's best results after optimization, in a secure environment

Optimization	Inference Time (s)	Recall (%)	Precision (%)	F1 (%)
None	4.76	99.9	90.3	94.9
Rem. TLS rec.	0.16	99.9	91.0	95.3
FP16	0.09	99.9	90.0	94.8

TABLE 5.6: C2 traffic detector's best results after optimization, in a secure environment

Both optimization techniques, of utilizing the half-precision floating-point format and eliminating all unnecessary TLS records, allowed us to remove around four seconds for the two scenarios. In other words, the inference time needed by Torpedo was reduced by around 9%, when the half-precision floating-point format was applied. Meanwhile, the time needed by the C2 detector was reduced by around 97%, when the TLS records were removed, and 98%, when the half-precision floating-point format was used.

When it comes to Torpedo, four seconds is not a big difference, since the inference time is still over 40 seconds. However, this time reduction permits the C2 scenario to go from 4.76 seconds to 0.16 and even 0.09 seconds, which is a massive difference by being

almost instantaneous. Additionally, this time improvement does not negatively affect the ML classification metrics. This is especially true for Torpedo since all classification metrics improved significantly.

Chapter 6

Conclusions

As industries grow and develop, their everyday needs also require more complex Machine Learning (ML) solutions. However, these days, security and privacy are just as important to an industry to progress and evolve. But, it was not until very recently that concern for ML security has been increasing. Perhaps, this is due to the fact that most of its procedures handle large amounts of private and sensitive data.

In 2020, BIML published a report identifying 78 security risks in ML systems today [48]. Out of the top ten risks mentioned in the report, we have data confidentiality, data trustworthiness, and output integrity. By applying Privacy-Preserving Machine Learning (PPML) and keeping ML predictors and the data used by them confidential, we can then mitigate these risks.

Even so, simply implementing PPML techniques, such as Secure Multiparty Computation (MPC) protocols, will not lead to an efficient final solution. It is very computationally expensive to preserve the privacy of an already complex model alongside a large quantity of data. Thus, in this work, we studied some optimization techniques that can be applied to network traffic Neural Networks (NN) to improve their performance in a secure and distributed environment.

This work describes the implementation of a ML-specific MPC framework, known as TF-Encrypted, and how it was applied to two network traffic application scenarios, so that their privacy can be guaranteed. The scenarios are named Torpedo and the C2 traffic detector and both make predictions about traffic flows. As for the distributed system, we utilized five Virtual Machines (VM), connected via Ethernet, to act as the MPC protocol participants. Only after configuring the MPC framework and the distributed system, did

we establish the baseline values of both scenarios. These values were then used to demonstrate if the optimizations mentioned in this work improved the NN's performance or not.

6.1 Future Work

During the development of this project, we encountered some obstacles and limitations that stopped us from perfecting our solution. Therefore, the following suggestions are merely guidelines for possible future works:

- Establish baseline values using a more realistic distributed system.
  - As mentioned previously, the VMs that constitute the distributed system are all connected via Ethernet. This means that the obtained baseline values are not as close to reality as intended, since Ethernet tends to offer faster connections than the Internet. So, to calculate the actual computation weights of a real distributed system, we need to connect the VMs using an Internet connection.
- Implement a general-purpose MPC framework in conjunction with the ML-specific protocol.
  - Due to time constraints, it was not possible to implement SCALE-MAMBA, a general-purpose MPC tool, together with TF-Encrypted. The original idea was to have SCALE-MAMBA perform some data pre-processing by filtering out some input to then use TF-Encrypted to execute a secure inference on the data. This was another possibility to optimize the NNs by reducing the amount of used data.
- Analyze more neural network optimization techniques.
  - There are a lot more NN optimization techniques available. However, given the complexity of implementing a MPC protocol, investigating said optimization techniques, testing them out and training the NN again and again, it was not possible to analyze every single one. But, given more time, we could take a look at parameter search and parameter decomposition techniques. Parameter search consists of searching for the best NN model among all possible different models. This is quite a difficult optimization to implement since the search

space can become ridiculously large. On the other hand, parameter decomposition fragments a vast weight matrix or tensor into smaller pieces to reduce the information loss as much as possible. However, it is very difficult, as well as time consuming, to find the right decomposition rank for a given tensor.

- Develop an optimization tool for network traffic machine learning.
  - After investigating a bigger set of optimization techniques, it should be possible to develop a tool for applying said techniques after selecting them. This would be extremely beneficial for network traffic ML engineers that wish to optimize their NN model to possibly run it in a secure environment.

Bibliography

[1] K. Sahinbas and F. Ö. Çatak, "Secure multi-party computation based privacy preserving data analysis in healthcare iot systems," *CoRR*, vol. abs/2109.14334, 2021. [Online]. Available: https://arxiv.org/abs/2109.14334 [Cited on pages 1 and 27.]

[2] D. Byrd and A. Polychroniadou, "Differentially private secure multi-party computation for federated learning in financial applications," in *Proceedings of the First ACM International Conference on AI in Finance*, ser. ICAIF '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3383455.3422562 [Cited on pages 1 and 27.]

[3] V. B. Kukkala, J. S. Saini, and S. Iyengar, "Secure multiparty computation of a social network," Ph.D. dissertation, Indian Institute of Technology Ropar, 2015. [Cited on pages 1 and 27.]

[4] "Towards responsible ai innovation: Second interpol-unicri report on artificial intelligence for law enforcement," May 2020. [Online]. Available: https://www.interpol.int/es/content/download/15290/file/AI%20Report%20INTERPOL%20UNICRI.pdf [Cited on page 1.]

[5] P. Medeiros, "Privacy-preserving deanonymization of tor circuits," Ph.D. dissertation, Instituto Superior Técnico da Universidade de Lisboa, 2020. [Cited on pages xiii, 3, 30, and 31.]

[6] C. Novo and R. Morla, "Flow-based detection and proxy-based evasion of encrypted malware c2 traffic," *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, 2020. [Cited on pages xiii, 3, 36, and 37.]

[7] C. Novo, J. M. Silva, and R. Morla, "An outlook on using packet sampling in flow-based c2 tls malware traffic detection," *2021 12th International Conference on Network of the Future (NoF)*, 2021. [Cited on pages 3 and 36.]

[8] K. Richards, "What is cryptography?" 9 2021. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/cryptography [Cited on page 5.]

[9] "Confidentiality." [Online]. Available: https://csrc.nist.gov/glossary/term/confidentiality [Cited on page 5.]

[10] "Integrity." [Online]. Available: https://csrc.nist.gov/glossary/term/integrity [Cited on page 5.]

[11] "Non-repudiation." [Online]. Available: https://csrc.nist.gov/glossary/term/non_repudiation [Cited on page 5.]

[12] "Authentication." [Online]. Available: https://csrc.nist.gov/glossary/term/authentication [Cited on page 5.]

[13] N. N. Kucherov, M. A. Deryabin, and M. G. Babenko, "Homomorphic encryption methods review," in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, 2020, pp. 370–373. [Cited on page 6.]

[14] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "Sok: General purpose compilers for secure multi-party computation," *2019 IEEE Symposium on Security and Privacy (SP)*, 2019. [Cited on pages xiii, 6, 8, 9, 10, 11, 17, 18, 19, and 21.]

[15] M. Lemus, M. F. Ramos, P. Yadav, N. A. Silva, N. J. Muga, A. Souto, N. Paunković, P. Mateus, and A. N. Pinto, "Generation and distribution of quantum oblivious keys for secure multiparty computation," *Applied Sciences*, 6 2020. [Cited on pages xiii and 7.]

[16] 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*. IEEE Computer Society, 1982, pp. 160–164. [Online]. Available: https://doi.org/10.1109/SFCS.1982.38 [Cited on page 7.]

[17] R. Tso, Z.-Y. Liu, and J.-H. Hsiao, "Distributed e-voting and e-bidding systems based on smart contract," *Electronics*, p. 422, 4 2019. [Cited on pages xiii and 8.]

[18] X. Wang, F. Xhafa, J. Ma, Y. Cao, and D. Tang, "Reusable garbled gates for new fully homomorphic encryption service," *International Journal of Web and Grid Services*, vol. 13, p. 25, 1 2017. [Cited on pages xiii and 9.]

[19] M. O. Rabin, "How to exchange secrets with oblivious transfer," 6 2005, harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005. [Online]. Available: http://eprint.iacr.org/2005/187 [Cited on page 9.]

[20] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1310–1321. [Online]. Available: https://doi.org/10.1145/2810103.2813687 [Cited on page 12.]

[21] "Ethics guidelines for trustworthy ai," 3 2021. [Online]. Available: https://ec.europa.eu/futurium/en/ai-alliance-consultation.1.html [Cited on page 13.]

[22] "A beginner's guide to neural networks and deep learning," 2020. [Online]. Available: https://wiki.pathmind.com/neural-network [Cited on pages xiii, 13, and 14.]

[23] B. Ramsundar and R. B. Zadeh, "Tensorflow for deep learning." [Online]. Available: https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html [Cited on page 14.]

[24] D. Oliver, B. Sick, and E. Murina, *Chapter 2: Neural Network Architectures*. Manning, 2020. [Cited on pages xiii and 14.]

[25] A. Varghese, "Convolutional neural networks," 7 2022. [Online]. Available: https://medium.com/data-science-community-srm/convolutional-neural-networks-5ec06c3223a [Cited on pages xiii and 15.]

[26] D. Bhatt, "Privacy-preserving in machine learning (ppml)," 2 2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2022/02/privacy-preserving-in-machine-learning-ppml/ [Cited on page 17.]

[27] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," *Computer Security - ESORICS 2008*, p. 192–206, 2008. [Cited on pages xiii, 17, 18, and 19.]

[28] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 643–662. [Cited on pages 17, 19, and 20.]

[29] "Repository for the scale-mamba mpc system." [Online]. Available: https://github.com/KULeuven-COSIC/SCALE-MAMBA [Cited on pages 17 and 20.]

[30] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, p. 26–49, 2019. [Cited on pages xiii, 17, and 22.]

[31] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "Flash: Fast and robust framework for privacy-preserving machine learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 2, p. 459–480, 2020. [Cited on pages 17 and 23.]

[32] 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*, vol. abs/1810.08130, 2018. [Online]. Available: http://arxiv.org/abs/1810.08130 [Cited on pages 17, 24, and 25.]

[33] C. Orlandi. (2021, 6) Mpc techniques series, part 9: Spdz. [Online]. Available: https://medium.com/partisia-blockchain/mpc-techniques-series-part-9-spdz-dbe1b7381e3b [Cited on page 20.]

[34] T. Araki, A. Barak, J. Furukawa, M. Keller, Y. Lindell, K. Ohara, and H. Tsuchida, "Generalizing the spdz compiler for other protocols," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 880–895. [Online]. Available: https://doi.org/10.1145/3243734.3243854 [Cited on pages xiii and 20.]

[35] Y. Dupis, "Encrypted deep learning training and predictions with tf encrypted keras," 8 2019. [Online]. Available: https://medium.com/dropoutlabs/encrypted-deep-learning-training-and-predictions-with-tf-encrypted-keras-557193284f44 [Cited on pages xiii, 24, and 25.]

[36] C. Hong, Z. Huang, W. Lu, H. Qu, L. Ma, M. Dahl, and J. Mancuso, "Privacy-preserving collaborative machine learning on genomic data using tensorflow," *CoRR*, vol. abs/2002.04344, 2020. [Online]. Available: https://arxiv.org/abs/2002.04344 [Cited on page 25.]

[37] M. Dahl, "Growing tf encrypted and officially becoming a community project," 5 2019. [Online]. Available: https://mortendahl.github.io/2019/05/17/growing-tf-encrypted/ [Cited on pages xiii and 26.]

[38] "The tor project: Privacy & freedom online." [Online]. Available: https://www.torproject.org/ [Cited on page 29.]

[39] "Tor project: How do onion services work?" [Online]. Available: https://community.torproject.org/onion-services/overview/ [Cited on page 30.]

[40] "Tor project: Research safety board." [Online]. Available: https://research.torproject.org/safetyboard/ [Cited on page 33.]

[41] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014. [Online]. Available: https://arxiv.org/abs/1412.6572 [Cited on pages xiii and 35.]

[42] [Online]. Available: https://malware-traffic-analysis.net/ [Cited on page 36.]

[43] M. Trevisan, A. Finamore, M. Mellia, M. Munafo, and D. Rossi, "Traffic analysis with off-the-shelf hardware: Challenges and lessons learned," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 163–169, 2017. [Cited on page 37.]

[44] "Tf-encrypted: A framework for encrypted machine learning in tensorflow." [Online]. Available: https://github.com/tf-encrypted/tf-encrypted.git [Cited on page 42.]

[45] S. Patel, A. Gupta, Nikhil, S. Kumari, M. Singh, and V. Sharma, "Network traffic classification analysis using machine learning algorithms," *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, 2018. [Cited on pages 50 and 52.]

[46] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *CoRR*, vol. abs/1502.02551, 2015. [Online]. Available: http://arxiv.org/abs/1502.02551 [Cited on page 51.]

[47] A. Murthy, H. Das, and M. A. Islam, "Robustness of neural networks to parameter quantization," *CoRR*, vol. abs/1903.10672, 2019. [Online]. Available: http://arxiv.org/abs/1903.10672 [Cited on page 51.]

[48] G. McGraw, H. Figueroa, V. Shepardson, and R. Bonett, "An architectural risk analysis of machine learning systems: Toward more secure machine learning," *Berryville Institute of Machine Learning*, 1 2020. [Cited on page 57.]