# ProteiNN: Privacy-preserving one-to-many Neural Network classifications

Beyza Bozdemir, Orhan Ermis, and Melek Önen
*EURECOM, Sophia Antipolis, France*
{*Beyza.Bozdemir, Orhan.Ermis, Melek.Onen*}*@eurecom.fr*

Keywords: privacy, neural networks, homomorphic proxy re-encryption

Abstract: In this work, we propose ProteiNN, a privacy-preserving neural network classification solution in a one-to-many scenario whereby *one* model provider outsources a machine learning model to the cloud server for its *many* different customers, and wishes to keep the model confidential while controlling its use. On the other hand, these customers take advantage of this machine learning model without revealing their sensitive inputs and the corresponding results. The solution employs homomorphic proxy re-encryption and a simple additive encryption to ensure the privacy of customers' inputs and results against the model provider and the cloud server, and to give the control on the privacy and use of the model to the model provider. A detailed security analysis considering potential collusions among different players is provided.

## 1 INTRODUCTION

The rapid evolution and utilisation of the Internet of Things (IoT) and cloud computing technologies results on the collection of large amounts of data. Companies inundated with such data can use machine learning (ML) techniques to learn more about their customers and hence improve their businesses. Nevertheless, the data being processed are usually privacy sensitive. Given the recent data breach scandals[1], companies face increasing challenges with ensuring data privacy guarantees and compliance with the General Data Protection Regulation (GDPR) while trying to bring value out of them. Furthermore, in addition to the processed (usually personal) data, companies that actually created/computed the machine learning models, may also wish not to make these publicly available. By hosting their machine learning models to cloud servers, companies are worried about losing their intellectual property.

We consider a scenario whereby a *model provider* owns a ML model and wishes to sell the use of this model to its customers that we name *queriers*. These queriers would like to obtain the classification/prediction of their inputs using the actual ML model. We assume that the model is outsourced to a *cloud server* who will be responsible for applying this model over queriers' inputs. As previously mentioned, both the ML model and the queriers' inputs are considered confidential. All parties are assumed to be potential adversaries: The model provider does not want to disclose its model to any party including the cloud server and the queriers; Similarly, a querier does not want to disclose its input and the result to any party including the cloud server and the model provider.

In this work, we focus on neural networks (NN). The problem of privacy-preserving NN classification has already been studied by many researchers (see (Azraoui et al., 2019) for a state of the art). Most of these works consider that the party who performs the NN operations is the model provider and is sufficiently powerful. On the other hand, (Hesamifard et al., 2018) and (Jiang et al., 2018) allow the outsourcing of these operations to the cloud server but the only querier, in this case, is the model provider. In both cases, the goal is to perform NN operations over queriers' inputs without leaking any information including the model. We propose to extend this scenario that we name the *one-to-one scenario*, by enabling *one* model provider to securely outsource its model to a cloud server and consider a *one-to-many scenario* whereby different customers can query the model. Additionally, while delegating the NN operations to the cloud server, the model provider also wishes to maintain the control over the use of this model by legitimate and authorised queriers, only.

To cope with the previously mentioned chal-

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

lenges, we propose ProteiNN, a privacy-preserving one-to-many NN classification solution that uses homomorphic proxy re-encryption. A homomorphic proxy re-encryption (H-PRE) scheme is a public-key encryption scheme that, on the one hand, thanks to its homomorphic property, enables operations over encrypted data, and, on the other hand, allows a third-party proxy to transform ciphertexts encrypted with one public key into ciphertexts encrypted with another public key without leaking any information about the underlying plaintext. With such a technique, the third-party cloud server can easily serve multiple different queriers without having access to the inputs, the results, and the model, and only the destined querier can decrypt the result. ProteiNN also makes use of a simple additive encryption scheme in order to let the model provider keep control over the queriers and to prevent potential collusions among ProteiNN parties. Our contributions can be summarized as follows:

- We develop ProteiNN which enables a model provider to delegate NN operations to a cloud server to serve multiple queriers while keeping control on the model use. As previously mentioned, ProteiNN combines H-PRE with a simple additive encryption in order to ensure the confidentiality of the model, the queriers' inputs, and their results. Only authorized queriers can decrypt their results.
- We consider potential collusions among any pair of players at the very design phase, namely: collusions between the cloud server and queriers, collusions between the cloud server and the model provider, and collusions between queriers and the model provider. We show that ProteiNN is collusion-resistant under the honest-but-curious security model.
- We implement ProteiNN as a proof-of-concept using the PALISADE library[2] and evaluate its performance with a particular NN model used for heart arrhythmia detection (Mansouri et al., 2019).

**Outline** Section 2 introduces the problem of privacy-preserving NN in a *one-to-many scenario* and describes the threat model. H-PRE is formally defined in Section 3. Section 4 describes ProteiNN in details. The security and performance evaluation of ProteiNN is provided in Section 5 and Section 6, respectively.

[2]https://git.njit.edu/palisade/PALISADE

# 2 PROBLEM STATEMENT

The literature features a number of privacy-preserving neural network (NN) solutions that enable queriers to request NN predictions without revealing their inputs and results to the model provider. Yet, these solutions are suitable to a specific setting whereby either the model provider is powerful and performs all operations on its side or, if the model provider outsources its operations, the only one who can query the model later on, is itself. In this section, we first overview the existing solutions, and further define our new setting and the threat model.

## 2.1 Prior work

A neural network (NN) is a layered machine learning technique consisting of interconnected processing units called *neurons* that compute specific functions in each layer. The first and last layers are defined as the *input* and the *output layers*, respectively. Each intermediate layer, named *hidden layers*, evaluates a function over the output of the previous layer and the result becomes the input to the next layer. These hidden layers usually consist of either linear operations such as matrix multiplications (fully connected or convolutional layers) or more complex operations such as sigmoid or max computation (activation layers or pooling layers). The reader can refer to (Tillem et al., 2020) for more information on the operations for each NN layer.

Performing neural network operations over confidential data requires the use of advanced privacy enhancing technologies (PETs) such as homomorphic encryption (HE) or secure multi-party computation (SMC) that unfortunately incur a non-negligible overhead. To efficiently integrate these PETs with neural networks, the design of the latter needs to be revisited. In particular, complex operations should be approximated to operations that can be efficiently supported by these PETs (such as low degree polynomials). Because such approximations have an impact on the accuracy of the underlying NN model, the goal of existing solutions mainly consists of addressing the trade-off between privacy, efficiency, and accuracy. Solutions either use HE (Gilad-Bachrach et al., 2016; Chabanne et al., 2017; Hesamifard et al., 2017; Ibarrondo and Önen, 2018; Bourse et al., 2018; Sanyal et al., 2018; Hesamifard et al., 2018; Jiang et al., 2018; Chou et al., 2018) or SMC (Mohassel and Zhang, 2017; Liu et al., 2017; Mohassel and Rindal, 2018; Rouhani et al., 2018; Riazi et al., 2018; Dahl et al., 2018; Wagh et al., 2019; Mansouri et al., 2019), or the combination of both (Barni et al., 2006; Orlandi et al.,

2007; Juvekar et al., 2018). Yet, most of these solutions do not take advantage of the cloud computing technology unless the cloud server has access to the model. Few solutions (Hesamifard et al., 2018; Jiang et al., 2018) rely on the existence of a cloud server and propose the idea of training NNs over encrypted data and classifying encrypted inputs: The model provider supplies the training data encrypted with its public key and the server builds the model. Nevertheless, during the classification phase, because the model is encrypted with the model provider's public key, the only party who can make queries (encrypted with the same public key) and have access to the classification results, is the model provider. Hence, other potential customers cannot query the model.

We propose to extend this one-to-one scenario to a one-to-many scenario and consider the case whereby a model provider outsources the encrypted NN model to a cloud server and many customers/queriers can issue classification requests without revealing the queries and the corresponding results.

## 2.2 Environment

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

More formally, our scenario illustrated in Figure 1 involves the following three players:

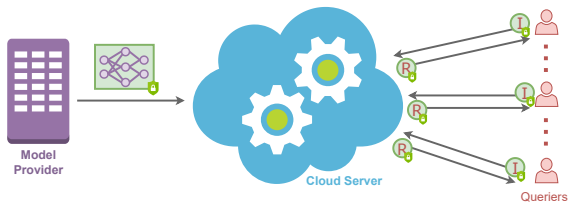


Figure 1: Players in the one-to-many scenario

- **The model provider** (MP) who owns a NN model (**M**): MP outsources encrypted **M** to an untrusted cloud server. MP wishes to have control over queriers who are willing to classify their input.

- **Querier** ($Q_i$) who queries the encrypted model **M**: Each querier $Q_i$ encrypts its input **I**. Later on, $Q_i$ receives the corresponding result **R** and can decrypt it correctly only if authorised by MP.

- **The cloud server** (CS) who stores the encrypted model **M** received from MP and performs the delegated NN operations over encrypted inputs received from different queriers.

## 2.3 Threat Model

Our threat model differs from the previous ones since the NN model is unknown to CS and we also consider potential collusion attacks. More specifically, we assume that all potential adversaries are honest-but-curious, i.e., parties adhere to the protocol steps but try to obtain some information about the model, the input or the result. Moreover, given the one-to-many setting and the introduction of the additional cloud server, we assume that collusions between CS and $Q_i$, between CS and MP, and between $Q_i$ and MP may exist. In this threat model, queriers aim at keeping their input **I** and the corresponding result **R** secret from CS and MP. On the other hand, MP does not want to disclose **M** to CS and $Q_i$. To summarize, ProteiNN considers the following potential adversaries: (i) An external adversary (who does not participate in ProteiNN) who should not learn any information about model **M**, input **I**, and result **R**; (ii) $Q_i$ who should not learn any information about model **M** even if $Q_i$ and CS collude[3]; (iii) CS who should not discover model **M**[4], input **I**, and the corresponding result **R** even if CS colludes with MP or querier(s); (iv) MP who should not learn anything about input **I** and its result **R** even when it colludes with CS or querier(s).

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

[3] Similar to previous works, we omit the attacks whereby $Q_i$ can try to re-build the model based on the authorised results that it receives.

[4] apart from its architecture

# 3 HOMOMORPHIC PROXY RE-ENCRYPTION

To cope with the one-to-many scenario, ProteiNN builds up on a homomorphic proxy re-encryption scheme which allows operations over encrypted data even when these are encrypted with different keys. In this section, we provide the formal definition of a homomorphic proxy re-encryption and briefly present the existing schemes. Table 1 sums up the notation used throughout the paper.

Table 1: Notations

| Symbol | Explanation |
|---|---|
| $\lambda$ | Security parameter |
| $N$ | Plaintext modulus |
| $(\mathsf{pk}_i, \mathsf{sk}_i)$ | Public-secret keys of party $i$ |
| $\mathsf{rek}_{i\to j}$ | Re-encryption key from party $i$ to party $j$ |
| $n$ | Number of queriers |
| $l$ | Number of queries from the querier $Q_i$ |
| $a \in_R A$ | $a$ is chosen randomly from $A$ |
| $r_{ij}$, $r'_{ij}$, $s_{ij}$, and $s'_{ij}$ | Randomly generated vectors in $\mathbb{Z}_N$ |
| $[\cdot]_{pk}$ | H-PRE encryption with public key pk |

## 3.1 Formal definition

As its name indicates, a Homomorphic Proxy Re-encryption (H-PRE) is the combination of two cryptographic constructions: (i) homomorphic encryption (HE) which enables operations over encrypted data (e.g. (Gentry, 2009b; Fan and Vercauteren, 2012; Halevi et al., 2018)); (ii) proxy re-encryption (PRE) which allows a third-party proxy such as a cloud server to transform ciphertexts encrypted with a public key into ciphertexts encrypted with another public key without learning any information on the plaintext ((Derler et al., 2018) provides an overview of existing PRE schemes).

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

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$: This probabilistic key generation algorithm takes the security parameter $\lambda$ as input, and outputs a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{rek}_{i\to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j)$: This re-encryption key generation algorithm takes secret key $\mathsf{sk}_i$ and public key $\mathsf{pk}_j$ as inputs and returns a re-encryption key, $\mathsf{rek}_{i\to j}$.
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$: This randomized algorithm encrypts message $m$ using public key pk and returns the resulting ciphertext $c$.
- $c \leftarrow \mathsf{Eval}(f, c^{(1)}, \cdots, c^{(t)})$: Given function $f$ and ciphertexts $c^{(1)}, \cdots, c^{(t)}$ where $c^{(i)} = \mathsf{Enc}(m^{(i)}, \mathsf{pk})$, the algorithm outputs ciphertext $c$ which corresponds to the encrypted evaluation of f. In general, a HE scheme has three separate methods for homomorphic addition, subtraction, and multiplication called EvalAdd, EvalSubt, and EvalMult, respectively.
- $c_j \leftarrow \mathsf{ReEncrypt}(\mathsf{rek}_{i\to j}, c_i)$: This re-encryption algorithm transforms a ciphertext $c_i$ into ciphertext $c_j$ using re-encryption key $\mathsf{rek}_{i\to j}$.
- $m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$: This algorithm decrypts the received ciphertext $c$ using secret key sk and outputs plaintext $m$.

According to (Gentry, 2009a; Halevi, 2017), any HE scheme can be transformed into a H-PRE scheme and its correctness implies the correctness of the resulting H-PRE. Furthermore, the same studies show that a H-PRE scheme is semantically secure (IND-CPA secure) if the underlying HE scheme is semantically secure.

## 3.2 Existing H-PRE solutions

The most relevant H-PRE schemes to ProteiNN are described in (Ding et al., 2017) and (Polyakov et al., 2017). Nevertheless, neither solution considers collusion attacks. Moreover, in (Polyakov et al., 2017), the publisher, who could be considered as a model provider in our scenario, does not have any control on subscribers. Finally, model $\mathbf{M}$ has to be re-encrypted as many times as the number of subscribers. In ProteiNN, we aim at extending the H-PRE scheme from (Polyakov et al., 2017) while ensuring all the privacy requirements defined in Section 2. Our threat model also takes collusion attacks into account and only authorised queriers can receive and decrypt the results of their classification queries.

# 4 PROTEINN

We consider the following two main problems arose by the one-to-many scenario: (i) each party uses a different public key to encrypt its data (model for MP and inputs for $\mathsf{Q}_i$) and (ii) queries received from $\mathsf{Q}_i$ should only be processed if these are authorised by MP. This setting implies that both the model and the queries should be encrypted with the same key at the classification step. With this aim, we introduce a Trusted Third Party (TTP) and use its public key as the common encryption key for both the model and the queries. TTP is considered as being offline: It does not play any role during the classification phase; it only distributes keying materials. H-PRE is used

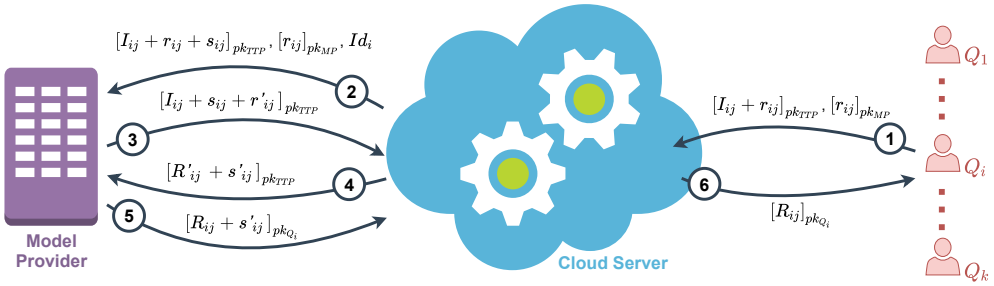


Figure 2: ProteiNN - Classification phase

towards the end of the classification phase, i.e., when $\mathsf{Q}_i$ needs to decrypt the actual result: Indeed, the result encrypted with TTP's public key needs to be re-encrypted with the $\mathsf{Q}_i$'s public key.

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

## 4.1 Setup Phase

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

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

1. Given security parameter $\lambda$, TTP and $\mathsf{Q}_i$ ($1 \leq i \leq n$) execute KeyGen and respectively obtain $(\mathsf{pk}_{\mathsf{TTP}}, \mathsf{sk}_{\mathsf{TTP}})$ and $(\mathsf{pk}_{\mathsf{Q}_i}, \mathsf{sk}_{\mathsf{Q}_i})$.
2. TTP generates re-encryption keys $\mathsf{rek}_{\mathsf{TTP}\to\mathsf{Q}_i} = \mathsf{ReKeyGen}(\mathsf{sk}_{\mathsf{TTP}}, \mathsf{pk}_{\mathsf{Q}_i}), \forall\ 1 \leq i \leq n$.
3. MP encrypts $\mathbf{M}$ with $\mathsf{pk}_{\mathsf{TTP}}$, and sends it to CS.
4. MP also generates random vectors $r_{ij}$ and $r'_{ij}$ for $\mathsf{Q}_i$ where $1 \leq j \leq l$ is the query number. MP encrypts $r_{ij}$ with $\mathsf{pk}_{\mathsf{TTP}}$., and stores them locally. Further, $[r_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}$ is sent to $\mathsf{Q}_i$. Finally, MP computes $\mathbf{M} * r'_{ij}$ for each classification, encrypts the results of $\mathbf{M} * r'_{ij}$ with $\mathsf{pk}_{\mathsf{TTP}}$, and stores them locally.

**Remark.** *Let* $\mathbf{M}$ *be the NN model and* $\mathbf{I}$ *be the input to be classified using* $\mathbf{M}$. $(*)$ *denotes the application of model* $\mathbf{M}$ *to input* $\mathbf{I}$*, and results in* $\mathbf{R} = \mathbf{M} * \mathbf{I}$. $(*)$ *consists of a combination of low-degree polynomial operations.*

## 4.2 Classification Phase

The classification phase of ProteiNN that is described below is illustrated in Figure 2. As previously mentioned, TTP is not involved in this phase.

1. $\mathsf{Q}_i$ encrypts $\mathbf{I}_{ij}$ with $\mathsf{pk}_{\mathsf{TTP}}$ and randomises it with vector $[r_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}$ received from MP. The result is further sent to CS.

1.1. $[\mathbf{I}_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}} = \mathsf{Enc}(\mathsf{pk}_{\mathsf{TTP}}, \mathbf{I}_{ij})$.

1.2. $[\mathbf{I}_{ij} + r_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}} = \mathsf{EvalAdd}([\mathbf{I}_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}, [r_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}})$.

2. CS randomises the value received in Step 1.2 and forwards the result to MP together with $[r_{ij}]_{\mathsf{pk}_{\mathsf{MP}}}$ and the identifier of the querier ($Id_i$).

2.1. $[\mathbf{I}_{ij} + r_{ij} + s_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}$
$= \mathsf{EvalAdd}([\mathbf{I}_{ij} + r_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}, [s_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}})$.

3. If $\mathsf{Q}_i$ is authorised[5], MP performs the following homomorphic operations over the received query and sends the outcome to CS.

3.1. $[\mathbf{I}_{ij} + s_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}$
$= \mathsf{EvalSubt}([\mathbf{I}_{ij} + r_{ij} + s_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}, [r_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}})$.

3.2. $[\mathbf{I}_{ij} + s_{ij} + r'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}$
$= \mathsf{EvalAdd}([\mathbf{I}_{ij} + s_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}, [r'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}})$.

4. CS subtracts $[s_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}$ from the value received in Step 3.4, performs the classification, randomises the result once again, and sends this value to MP.

4.1. $[\mathbf{I}_{ij} + r'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}$
$= \mathsf{EvalSubt}([\mathbf{I}_{ij} + s_{ij} + r'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}, [s_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}})$.

4.2. $[\mathbf{R}'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}} = [\mathbf{M}]_{\mathsf{pk}_{\mathsf{TTP}}} * [\mathbf{I}_{ij} + r'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}$.

4.3. $[\mathbf{R}'_{ij} + s'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}$
$= \mathsf{EvalAdd}([\mathbf{R}'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}, [s'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}})$.

[5] If/whenever MP does not want to authorise querier $\mathsf{Q}_i$, MP can send a reject message to CS, and thus, CS would terminate the protocol.

5. MP re-encrypts the received value with $\mathsf{rek}_{\mathsf{TTP}\to\mathsf{Q}_i}$ and sends it to CS[6].

5.1. $[\mathbf{R}_{ij}+s'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}$
$= \mathsf{EvalSubt}([\mathbf{R'}_{ij}+s'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}}, [\mathbf{M}*r'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}})$.

5.2. $[\mathbf{R}_{ij}+s'_{ij}]_{\mathsf{pk}_{\mathsf{Q}_i}}$
$= \mathsf{ReEncrypt}(\mathsf{rek}_{\mathsf{TTP}\to\mathsf{Q}_i}, [\mathbf{R}_{ij}+s'_{ij}]_{\mathsf{pk}_{\mathsf{TTP}}})$ .

6. CS homomorphically removes $s'_{ij} \in_R \mathbb{Z}_N$ from this value and sends the result to $\mathsf{Q}_i$.

6.1. $[\mathbf{R}_{ij}]_{\mathsf{pk}_{\mathsf{Q}_i}} = \mathsf{EvalSubt}([\mathbf{R}_{ij}+s'_{ij}]_{\mathsf{pk}_{\mathsf{Q}_i}}, [s'_{ij}]_{\mathsf{pk}_{\mathsf{Q}_i}})$.

7. $\mathsf{Q}_i$ decrypts this value with its private key in order to obtain the classification result $\mathbf{R}_{ij}$.

7.1. $\mathbf{R}_{ij} = \mathsf{Dec}(\mathsf{sk}_{\mathsf{Q}_i}, [\mathbf{R}_{ij}]_{\mathsf{pk}_{\mathsf{Q}_i}})$.

# 5 SECURITY ANALYSIS

We analyze the security of ProteiNN considering our newly introduced threat model and show that it satisfies the privacy requirements defined in Section 2. We propose to conduct this security analysis incrementally, by taking each adversary into account, one-by-one, and the potential collusions. As described in Section 2, all ProteiNN parties are considered as honest-but-curious adversaries. Furthermore, we assume that the H-PRE scheme that ProteiNN uses is semantically secure and that the encrypted addition of random vectors $r_{ij}$, $r'_{ij}$, $s_{ij}$, and $s'_{ij}$ is considered as a perfectly secure.

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

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

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

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

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

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

**$\mathsf{Q}_i$-MP collusions.** Collusions between $\mathsf{Q}_i$ and MP do not result in any leakage regarding other queriers' inputs and results thanks to the randomization of both input and result. More precisely, even if malicious $\mathsf{Q}_i$ and MP collude to discover **I** and **R** of legitimate $\mathsf{Q}_t$, they cannot retrieve them because of the use of random vectors $s_{tj}$ and $s'_{tj}$.

***Note on $\mathsf{Q}_i$-MP-CS collusions.*** We consider that all three players cannot collude since in this case there is no need for privacy protection.

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

# 6 PERFORMANCE EVALUATION

We propose to evaluate the performance of ProteiNN using an arrhythmia detection case study described in (Mansouri et al., 2019) whereby MP owns a NN model for the classification of heart arrhythmia; Queriers' inputs consist of the individuals' ElectroCardioGram (ECG) data and the result is the actual arrhythmia type the patient suffers from. The

[6]In case some revocation of $\mathsf{Q}_i$ occurs, MP has, once again, the opportunity to reject the query and will not do any re-encryption.

underlying NN model has been built in (Mansouri et al., 2019). It consists of two fully connected layers and one activation layer implemented with the square function. It involves 16 input neurons, 38 hidden neurons, and 16 output neurons. The model provides an accuracy of 96.34% (see Section 3.3 in (Mansouri et al., 2019)).

**Experimental setup.** To implement ProteiNN, we have utilised the PALISADE library (v1.5.0) supports several HE schemes and their PRE versions. The H-PRE scheme we employ for ProteiNN is H-BFVrns (Halevi et al., 2018) mainly because it is the most efficient in PALISADE. We follow the standard HE security recommendations (e.g., 128-bit security) indicated in (Chase et al., 2017) for H-BFVrns. The ProteiNN steps for TTP and CS were carried out using a desktop computer with 4.0 GHz Intel Core i7-7800X processor, 128 GB RAM, and the Ubuntu 18.04.3 LTS operating system whereas the steps for $\mathsf{Q}_i$ and MP were performed using a laptop with 1.8 GHz Intel Core i7-8550U processor, 32 GB RAM, and the Ubuntu 18.10 operating system.

Table 2: Performance results for ProteiNN

| **Setup phase** | | |
|---|---|---|
| **ProteiNN Step** | **Player** | **Time (ms)** |
| Step 1 - Key generation | TTP | 20.02 |
| Step 1 - Key generation | $\mathsf{Q}_i$ | 21.17 |
| Step 2 - Re-encryption key generation | TTP | 267.57 |
| Step 3 - Model encryption | MP | 1514.08 |
| Step 4 - Random number generation, $M * r'_{ij}$ & encryption | MP | 63.47 |
| **Classification phase of one input** | | |
| **ProteiNN Step** | **Player** | **Time (ms)** |
| Step 1.1 - Input encryption | $\mathsf{Q}_i$ | 31.72 |
| Step 1.2 - Random addition to Input | $\mathsf{Q}_i$ | 1.95 |
| Step 2.1 - Input randomisation with random generation | CS | 31.73 |
| Step 3.1 - Randomisation removal from the Input | MP | 1.96 |
| Step 3.2 - Random addition to Input | MP | 1.95 |
| Step 4.1 - Randomisation removal from the Input | CS | 1.77 |
| Step 4.2 - Classification | CS | 26183.5 |
| Step 4.3 - Result randomisation with random generation | CS | 31.78 |
| Step 5.1 - $[\mathbf{M} * r]_{\mathsf{pk}_{\mathsf{TTP}}}$ removal from the Result | MP | 1.92 |
| Step 5.2 - Result re-encryption | MP | 30.12 |
| Step 6.1 - Randomisation removal from the Result | CS | 1.84 |
| Step 7.1 - Result decryption | $\mathsf{Q}_i$ | 5.57 |
| TOTAL | | 26325.81 |

We have evaluated the performance of both the setup and classification phases. Detailed results are depicted in Table 2. These results correspond to the average from the execution of 100 individual simulations. We observe that one ProteiNN classification instance takes 26.33 s, approximately. Only the cloud server performs costly operations. Indeed, the querier takes around 31 and 5 ms, to encrypt and decrypt its input and result, respectively. The cost of ReEncrypt (about 30 ms) seems negligible when compared to the cost of the classification phase. Among the operations performed by MP, the most costly one is the encryption of $\mathbf{M}$ (about 1.5 s). It is worth to note that this operation is performed during the Setup phase and only once. The other remaining operations performed by MP are in the order of 30 ms.

Table 3: Performance results for ProteiNN players

| **Setup phase** | |
|---|---|
| **ProteiNN Player** | **Time (ms)** |
| TTP | 287.59 |
| $\mathsf{Q}_i$ | 21.17 |
| MP | 1577.55 |
| **Classification phase** | |
| **ProteiNN Player** | **Time (ms)** |
| $\mathsf{Q}_i$ | 39.24 |
| MP | 35.95 |
| CS | 26250.62 |

As shown in Table 3, we observe that while CS takes 26.25 s to classify a heartbeat, MP and $\mathsf{Q}_i$ only take 36 and 39 ms, respectively. We have also evaluated the classification cost for MP in a one-to-one scenario in order to justify the need for cloud servers. In this context, MP has to compute all costly operations over the encrypted input and the cleartext model. With the same HE library, MP takes 1.81 s. In the one-to-many scenario, this cost is much smaller (only 36 ms). It is worth to note that when dealing with deeper neural networks, this gap will be much larger and hence the use of cloud servers is justified.

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

# 7 CONCLUSION

We have proposed ProteiNN, a privacy-preserving one-to-many NN classification solution that is based on the use of H-PRE and a simple additive encryption. ProteiNN achieves confidentiality for the model(s), the inputs, and the results. Additionally, the model provider also has control over the model outsourced the cloud server. We have provided a detailed security analysis by considering all potential adversaries including collusions among them. We have implemented ProteiNN as a proof-of-concept with a case study and our work shows promising performance results and calls for future work to evaluate the scalability of ProteiNN. We believe that with an appropriate batched classification and a powerful cloud server, ProteiNN could provide better performance and be scalable with respect to the number of queriers.

## ACKNOWLEDGEMENTS

We would like to thank Yuriy Polyakov and Yarkın Doröz for their valuable suggestions on the implementation. This work was partly supported by the PAPAYA project funded by the European Union's Horizon 2020 Research and Innovation Programme, under Grant Agreement no. 786767.


## REFERENCES


Azraoui, M., Bahram, M., Bozdemir, B., Canard, S., Ciceri, E., Ermis, O., Masalha, R., Mosconi, M., Önen, M., Paindavoine, M., Rozenberg, B., Vialla, B., and Vicini, S. (2019). SoK: Cryptography for neural network. In *IFIP Summer School on Privacy and Identity Management*.

Barni, M., Orlandi, C., and Piva, A. (2006). A privacy-preserving protocol for neural-network-based computation. In *Multimedia & Security*.

Bourse, F., Minelli, M., Minihold, M., and Paillier, P. (2018). Fast homomorphic evaluation of deep discretized neural networks. In *CRYPTO*.

Chabanne, H., de Wargny, A., Milgram, J., Morel, C., and Prouff, E. (2017). Privacy-preserving classification on deep neural network. *Cryptology ePrint Archive*.

Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Hoffstein, J., Lauter, K., Lokam, S., Moody, D., Morrison, T., Sahai, A., and Vaikuntanathan, V. (2017). Security of homomorphic encryption. Technical report, HomomorphicEncryption.org.

Chou, E., Beal, J., Levy, D., Yeung, S., Haque, A., and Fei-Fei, L. (2018). Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *CoRR*.

Dahl, M., Mancuso, J., Dupis, Y., Decoste, B., Giraud, M., Livingstone, I., Patriquin, J., and Uhma, G. (2018). Private machine learning in tensorflow using secure computation. *CoRR*.

Derler, D., Krenn, S., Lorünser, T., Ramacher, S., Slamanig, D., and Striecks, C. (2018). Revisiting proxy re-encryption: Forward secrecy, improved security, and applications. Cryptology ePrint Archive.

Ding, W., Yan, Z., and Deng, R. H. (2017). Encrypted data processing with homomorphic re-encryption. *Inf. Sci.*

Fan, J. and Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive.

Gentry, C. (2009a). A Fully Homomorphic Encryption Scheme.

Gentry, C. (2009b). Fully homomorphic encryption using ideal lattices. In *STOC*.

Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K. E., Naehrig, M., and Wernsing, J. (2016). Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*.

Halevi, S. (2017). Homomorphic encryption. *Tutorials on the Foundations of Cryptography, Information Security and Cryptography*.

Halevi, S., Polyakov, Y., and Shoup, V. (2018). An improved rns variant of the bfv homomorphic encryption scheme. Cryptology ePrint Archive.

Hesamifard, E., Takabi, H., and Ghasemi, M. (2017). Cryptodl: Deep neural networks over encrypted data. *CoRR*.

Hesamifard, E., Takabi, H., Ghasemi, M., and Wright, R. N. (2018). Privacy-preserving machine learning as a service. *PoPETs*.

Ibarrondo, A. and Önen, M. (2018). Fhe-compatible batch normalization for privacy preserving deep learning. In *DPM*.

Jiang, X., Kim, M., Lauter, K. E., and Song, Y. (2018). Secure outsourced matrix computation and application to neural networks. In *ACM CCS*.

Juvekar, C., Vaikuntanathan, V., and Chandrakasan, A. (2018). GAZELLE: A low latency framework for secure neural network inference. In *USENIX*.

Liu, J., Juuti, M., Lu, Y., and Asokan, N. (2017). Oblivious neural network predictions via minionn transformations. In *ACM CCS*.

Mansouri, M., Bozdemir, B., Önen, M., and Ermis, O. (2019). PAC: Privacy-preserving arrhythmia classification with neural networks. In *FPS*.

Mohassel, P. and Rindal, P. (2018). ABY$^3$: A mixed protocol framework for machine learning. In *ACM CCS*.

Mohassel, P. and Zhang, Y. (2017). SecureML: A system for scalable privacy-preserving machine learning. In *S&P*.

Orlandi, C., Piva, A., and Barni, M. (2007). Oblivious neural network computing via homomorphic encryption. *EURASIP*.

Polyakov, Y., Rohloff, K., Sahu, G., and Vaikuntanathan, V. (2017). Fast proxy re-encryption for publish/subscribe systems. *Trans. Priv. Secur.*

Riazi, M. S., Weinert, C., Tkachenko, O., Songhori, E. M., Schneider, T., and Koushanfar, F. (2018). Chameleon: A hybrid secure computation framework for machine learning applications. In *AsiaCCS*.

Rouhani, B. D., Riazi, M. S., and Koushanfar, F. (2018). Deepsecure: scalable provably-secure deep learning. In *DAC*.

Sanyal, A., Kusner, M. J., Gascón, A., and Kanade, V. (2018). TAPAS: tricks to accelerate (encrypted) prediction as a service. *CoRR*.

Tillem, G., Bozdemir, B., and Önen, M. (2020). SwaNN: Switching among cryptographic tools for privacy-preserving neural network predictions. In *preprint*.

Wagh, S., Gupta, D., and Chandran, N. (2019). SecureNN: Efficient and private neural network training. In *PoPETs*.