Contents

List of Tables

List of Figures

Nomenclature

BFV	Brakerski/Fan-Vercauteren
CNN	Convolutional Neural Network
ELM	Extreme Learning Machines
FHE	Fully Homomorphic Encryption
HE	Homomorphic Encryption
NB	Noise Budget
NTT	Number-Theoretic Transform
RLWE	Ring Learnig With Error
SEAL	Simple Encrypted Arithmetic Library
SHE	Somewhat Homomorphic Encryption

Chapter 1

Introduction

1.0.1 Problem and Motivations

1.0.1.1 Artificial Intelligence and Privacy

Artificial intelligence (AI) is the concept used to describe computer systems that are able to learn from their own experiences and solve complex problems in different situations – abilities we previously thought were unique to mankind. And it is data, in many cases personal data, that fuels these systems, enabling them to learn and become intelligent. The AI spring has dawned thanks to the availability of huge amounts of data, coupled with an increase in processing power and access to cheaper and greater storage capacity. Big Data often refers to vast volumes of data, extracted from multiple sources, often in real time. These enormous data streams can be utilized for the benefit of society by means of analysis and finding patterns and connections. Due to its potential, AI is starting to play a leading role in many different fields, such as: improve the efficiency of public sectors, provide new methods of climate and environmental protection, build a safer society, and perhaps even find a cure for cancer. Today we see that AI is used to solve specific tasks such as, for example, image and speech recognition.
Deep learning is a form of machine learning, whose mechanism of functioning is inspired by the human brain's and are focused on learning data representation as opposed to task-specific algorithms. Besides the many advantages that AI can bring, we should consider its privacy implications, that arise from the usage of Big Data. This is a crucial point since, if people cannot trust that information about them is being handled properly, it may limit their willingness to share information in their everyday life. If we find ourselves in a situation in which sections of the population refuse to share information because they feel that their personal integrity is being violated, we will be faced with major challenges to our freedom of speech and to people's trust in the authorities. A refusal to share personal information will also represent a considerable challenge with regard to the commercial use of such data in sectors such as the media, retail trade and finance services. The answer to the question as to whether it is possible to use

AI, and protect people's data while doing so, is yes. It is both possible and necessary in order to safeguard fundamental personal data protection rights.

1.0.1.2 The EU-GDPR

The necessity to take into account privacy when dealing with Artificial Intelligence has been strengthen by the entry into force on 24 May 2016 of the **EU's General Data Protection Regulation (GDPR)**, which, following a two year post-adoption grace period, has been directly applied in all EU Member States starting form 25 May 2018[1]. Organizations have less than a year to prepare for compliance. The GDPR replaces the Data Protection Directive 95/46/EC and was designed to harmonize data privacy laws across Europe, to protect and empower all EU citizens data privacy and to reshape the way organizations across the region approach data privacy. The GDPR has a broad territorial scope. It applies not only to all organizations established in the EU that handles personal data but also to any non-EU established organization that processes personal data of individuals who are in the EU in order to: offer them goods or services, irrespective of whether a payment is required; monitor their behavior within the EU. The GDPR aims to protect personal data at all stages of data processing and it identifies two different entities that both have obligations: data controllers and data processors.The provisions of the GDPR govern the data controller's duties and the rights of the data subject when personal information is processed. The GDPR therefore applies when artificial intelligence is *under development* with the help of personal data, and also when it is used to *analyze* or *reach decisions* about individuals.

Before going deeper into the details of the newness introduced, it is useful to make some clarifications about the terminology and the entities involved. First of all, it is important to clarify what we mean when we speak about *personal data*. Personal data means any information relating to an identified or identifiable natural person (GDPR Article 4 (1)). The data may be directly linked to a person, such as a name, identification number or location data. The data may also be indirectly linked to a person. This means that the person can be identified on the basis of a combination of one or more elements that are specific to a person's physical, physiological, genetic, mental, economic, cultural or social identity. *Sensitive data* is a special sub-category of personal data which holds extra consideration and protection in GDPR as they may give rise to strong stigmatization or discrimination in society; they include information about racial or ethnic origin, political convictions, religious or philosophical beliefs or trade union membership, as well as the processing of genetic and biometric data with the aim of uniquely identifying a natural person, health details or information regarding a person's sexual relationships or sexual orientation (GDPR Article 4). Secondly the *data controller* is the natural or legal person, public authority, agency or other body which, alone or jointly with others, determines the purposes and means of processing of personal data (GDPR Article 4 (7)), where *processing* means any operation or set of operations which is performed on personal data, such as collection, recording, organization, struc-

turing, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available, alignment or combination, restriction, erasure or destruction (GDPR Article 4 (2)).

The rules governing the processing of personal data have their basis in some fundamental principles. Article 5 of the GDPR lists the principles that apply to all personal data processing. The essence of these principles is that personal information shall be utilized in a way that protects the privacy of the data subject in the best possible way, and that each individual has the right to decide how his or her personal data is used. The use of personal data in the development of artificial intelligence challenges several of these principles. In summary, these principles require that personal data is:

- processed in a lawful, fair and transparent manner (principle of legality, fairness and transparency)
- collected for specific, expressly stated and justified purposes and not treated in a new way that is incompatible with these purposes (principle of purpose limitation)
- adequate, relevant and limited to what is necessary for fulfilling the purposes for which it is being processed (principle of data minimization)
- correct and, if necessary, updated (accuracy principle)
- not stored in identifiable form for longer periods than is necessary for the purposes (principle relating to data retention periods)
- processed in a way that ensures adequate personal data protection (principle of integrity and confidentiality)

In addition, the data controller is responsible for, and shall be able to prove, compliance with the principles (accountability principle).

The new data protection regulations will enhance the rights of individuals. At the same time, the duties of organizations will be tightened up. Two new requirements that are especially relevant for organizations using AI, are the requirements *privacy by design* and the *Data Protection Impact Assessment (DPIA)*. To enable privacy by design, the data controller shall build privacy protection into the system and ensure that data protection is safeguarded in the system's standard settings. These requirements are described in Article 25 of the GDPR and apply when developing software, ordering new systems, solutions and services, as well as when developing these further. The rules require that data protection is given due consideration in all stages of system development, in routines and in daily use. Standard settings shall be as protective of privacy as possible, and data protection features shall be embedded at the design stage. The principle of data minimization is expressly mentioned in the provision relating to privacy by design. *Data minimization* is a principle aiming to limit the amount of detail included in training or in the use of a model. This may be achieved by making it difficult to identify the individuals contained in

the basic data. The degree of identification is restricted by both the amount and the nature of the information used, as some details reveal more about a person than others. The use of pseudonymization or encryption techniques protect the data subject's identity and help limit the extent of intervention. This principle also forces developers to thoroughly examine the intended area of application of the model to facilitate selection of relevant data necessary for the purpose. Furthermore, the developer must consider how to achieve the objective in a way that is least invasive for the data subjects. The assessments performed need to be documented, so that they can be presented to the Data Protection Authority, that should monitor, in the event of an inspection, or in connection with a preliminary discussion. The Data Protection Impact Assessment (DPIA) must be conduct by anyone processing personal data, for example by an enterprise that believes that a planned process is likely to pose a high risk to natural persons' rights and freedoms. This is described in Article 35 of the GDPR. Moreover, there is a requirement to assess the impact on personal privacy by systematically and extensively considering all personal details in cases where this data is used in automated decision making, or when special categories of personal data (sensitive personal data) are used in on a large scale. The systematic and large-scale monitoring of public areas also requires documentation showing that a DPIA has been conducted. The impact assessment should include the following as a minimum:

- a systematic description of the process, its purpose and which justified interest it protects
- an assessment of whether the process is necessary and proportional, given its purpose
- an assessment of the risk that processing involves for people's rights, including the right to privacy
- the measures selected for managing risk identified

Information must also be provided regarding risks, rules, safeguards, and the rights of the data subjects in connection with processing, as well as how these rights can be exercised. In addition, an *extended duty to inform* will apply when personal data is collected for automated decision- making. The use of artificial intelligence is a form of automated processing, and, moreover, in some cases the decision is taken by the model.[5]

1.0.2 Objectives

This study focuses on Convolutional Neural Networks (CNNs) and presents both a methodology and a library to convert a learned neural network to a network that can be applied on encrypted data (Encoded Network), by employing a Homomorphic Encryption scheme. This allows a data owner to encrypt his data, with his public key, before to send them to a cloud service that hosts the

network. The encryption ensures that the data remain confidential since the cloud doesn't have access to the, secret key needed to decrypt the data, but at the same time, it is able to perform the computations required by the neural network on the encrypted data, to make encrypted predictions and to return them to the data owner in an encrypted form. Thus, only the data owner in the end will be able to decrypt the result and obtain his predictions. Therefore the cloud service does not gain any information about the raw data nor about the prediction it made.
Moreover, since the usage of HE decreases the performances of the model, in the sense that computations are much slower than computations done on plain data and require much more memory, the study proposes a mathematical formulation, to understand which is the optimal neural network, to convert in an Encoded Network, given some constraints for the encryption parameters and provide an heuristic algorithm to solve the problem stated by the given mathematical formulation .

1.0.3 Results

Chapter 2

State of the Art

This Chapter focuses on most recent studies on methods for Privacy-Preserving Machine Learning and on other related interesting applications of homomorphic encryption in different fields.
Most of these works are based on one of these two scenarios:

1. There are multiple data providers whose data must be commonly protected
2. There is a single data provider

In both cases their data are sent to an outsourced server which should make some computations but which is supposed to be a *curious- but-honest* server; i.e., the server carries out the result of computations faithfully, but it may try to learn as much as possible about the inputs and/or outputs of the computation. The processed data can be sent back to the data provider/s or to a third parti, a data analyst, that is allowed to decrypt the result.

2.1 Multiple Data Providers

A possible approach to preserve privacy while examining data on outsourced untrusted parti is to use secure *Multi-Party Computation (MPC)* techniques [25], which are focused on establishing a communication protocol between the parties involved, such that if the parties follow it they will end with the desired results while protecting the security and privacy of their respective assets.
Barni et al. (2016) in [6] propose an iterative method to allow the privacy preserving forward phase on a neural network. The data owner encrypts the data and sends it to the cloud. The cloud computes an inner product between the data and the weights of the first layer, and sends the result to the data owner. The data owner decrypts, applies the non-linear transformation, and encrypts the result before sending it back to the cloud. The cloud can apply the second layer and send the output back to the data owner. The process continues until all the layers have been computed. However in 2007 Orlandi et al. in [34] have shown that this process leaks information of the weights to the data owner and

therefore propose a method to obscure the weights.
Kuri et al. (2017) in [29] use additively homomorphic encryption in combination with Extreme Learning Machine (ELM). ELM is a feedforward neural network where connection weights from the input layer to the hidden layer are randomly generated and the connection weights from the hidden layer to the output layer are learned analytically [11]. In particular they focus on a single hidden layer feedforward neural network (SLFN) and provide both training and testing algorithm on homomorphically encrypted data. Due to its simple structure ELM has relatively fast learning speed and higher accuracy as a nonlinear classifier and since only one homomorphic addition is performed, the computational overhead added is quite small. On the other hand this simplicity does not allow to accomplish more complex tasks, such as object recognition in images as it is possible to do with CNNs.
Le Trieu Phong et al. (2017) in [36] propose a privacy-preserving deep learning system in which many learning participants perform neural network-based deep learning over a combined dataset of all, without actually revealing the participants' local data to a central server. They revisit the previous work [41] and point out that local data information may be actually leaked to an honest-but-curious server, then they move on to fix that problem via building an enhanced system that also keeps accuracy intact. Each learning participant is also a data provider and has a local copy of the neural network to train. The proposed system aims at training the weights utilizing multiple data sources and gradient-encrypted Asynchronous Stochastic Gradient Descent (ASGD). There is a global weight vector W_{global}, initialized randomly. At each iteration, replicas of the neural network (one for each participant), are run over local dataset, and the corresponding local gradient vector G_{local} is sent to the cloud. For each G_{local} , the cloud then updates the global parameters $W_{global} := W_{global} - \alpha \cdot G_{local}$ with α learning rate. The updated global parameters are broadcast to all the replicas, which then use them to replace their old weight parameters, until a minimum for a pre-defined cost function is reached. As showed in Figure 2.1, to ensure privacy, additively homomorphic encryption is used and the updating formula becomes $E(W_{global}) := E(W_{global}) + E(-\alpha \cdot G_{local})$ and to ensure integrity of the homomorphic ciphertext, each client uses a secure channel such as TLS/SSL to communicate the homomorphic ciphertexts to the server. To use the power of parallel computation when the server has multiple processing units $PU_1, \ldots, PU_{npu}$, ASGD splits the weight vector W and gradient vector G into n_{pu} parts, so that each processing unit PU_i computes the update rule $W_i = W_i - G_i$.
The participants jointly set up the public key pk and secret key sk for an additively homomorphic encryption scheme. The secret key sk is kept confidential against the cloud server, but is known to all learning participants. Each participant establishes a TLS/SSL secure channel, different from each other, to communicate and protect the integrity of the homomorphic ciphertexts. The downloads and uploads of the encrypted parts of W_{global} can be asynchronous in two aspects: the participants are independent with each other; and the processing units are also independent with each other. However protecting the

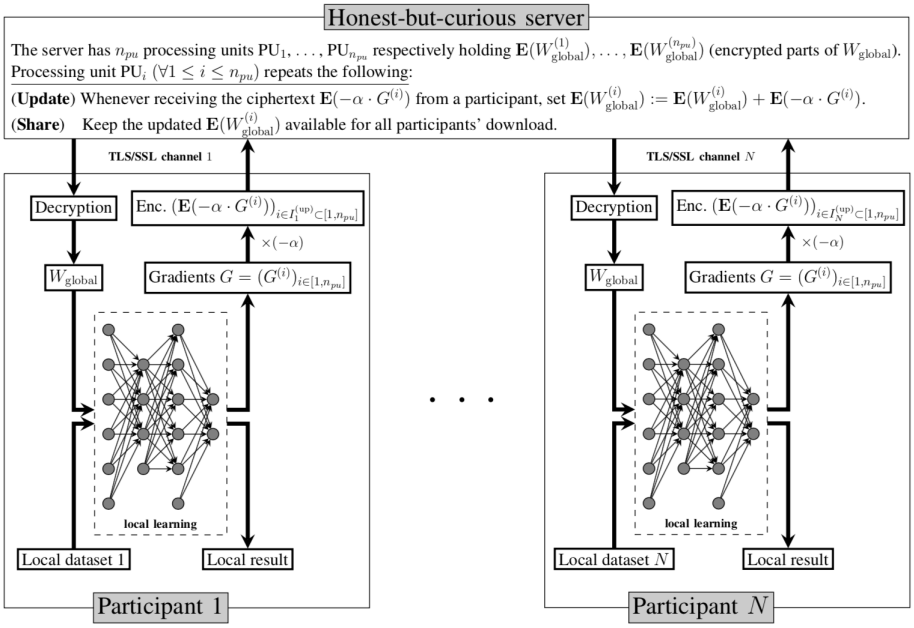


Figure 2.1: Gradients-encrypted Asynchronous SGD for privacy-preserving deep learning, with a curious cloud server and N honest participants. [36]

gradients against the cloud server comes with the cost of increased communication between the learning participants and the cloud server. Saeed Samet et al. (2012) in [40] focus their work on training ELM networks when data is vertically or horizontally distributed distributed among parties (data owners), using as sub-protocols secure multi-parti multiplication and secure multi-parti addition. The model (i.e. the network) is securely constructed and distributed among the parties involved. Indeed, at the end of the learning protocol the parties can jointly use the model on target data to predict the corresponding output. This seems to be a different approach with respect to the the two previous presented works ([29] , [36]) where there was always a central entity (server/cloud) designated to aggregate partial results coming form the different parties. They also prove that their solution is resistant to collusion attacks [39] since if n parties are involved in the protocol, and *n-1* of them collude to obtain the private information of the lest party, the only information they will be able to get is their input and output share. This information will not help them find the input and outputs shares of the last party, because they have n equations with at least *n+1* unknown values. Therefore, secure multi-party multiplication as well as secure multi-parti addition are secure against collusion attacks by up to $n-1$ parties and can be used over public channels. The problem with this protocol is that the time needed for training increases with the number of

parties involved.
All these works have in common the fact that they use relatively simple HE schemes, that is, additively or multiplicative HE that allow to perform homomorphically only one of these two operations. In those cases HE as RSA, that is multiplicative homomorphic or Paillier [35] that is additively homomorphic are usually used.

2.2 Single Data Provider

Graepel et al. (2013) [26] suggest a way to adapt machine learning algorithms, as classification, in order to train them over encrypted data. Therefore, for homomorphic encryption limitations, they are forced to use functions that learn low degree polynomials. As a result, most of the algorithms proposed are of the linear discrimination type and therefore for many tasks they do not deliver the same level of accuracy as neural networks are capable of delivering.
Yoshinori Aono et al, (2016) in [3] has proposed an homomorphism-aware logistic regression via approximation of the logarithm in the cost function to minimize, with a polynomial of degree k (e.g. k=2). Since HE ensures data secrecy, *differential privacy* is added to the model to ensure also output privacy.

Differential privacy is a technique that aims to maximize the accuracy of queries from statistical databases (here, the term statistical database means a set of data that are collected under the pledge of confidentiality for the purpose of producing statistics that, by their production, do not compromise the privacy of those individuals who provided the data) while measuring impact on individuals whose information is in the database. Intuitively it can be explained by saying that if there are two datasets D_1 and D_2 that differ on a single element (i.e., the data of one person), a given differentially private algorithm will behave approximately the same on both datasets. More formally,

Definition 1. ($\epsilon-$Differential Privacy [18]) A randomized mechanism M, that answers to queries on a database, provides ε-differential privacy, if, for all databases D_1 and D_2 which differ by at most one element, and for any t,

$$\frac{Pr[M(D_1) = t]}{Pr[M(D_2) = t]} \leq e^{\varepsilon}.$$

It has been shown in [17] that if a mechanism satisfies ε-differential privacy, then an adversary who knows the private value of all the individuals in the dataset, except for one single individual, cannot figure out the private value of the unknown individual, with sufficient confidence, from the responses of the mechanism M. ε-differential privacy is therefore a very strong notion of privacy [12].
In [3] to add ϵ-differential privacy they change the computation done at the server side. Namely, the server generates Laplace noises, encrypts and sums them to its computation. The result is that the data analyst (or data provider),

finds, after decryption, exactly the Laplace-perturbed coefficients of the cost function. Even if good results are obtained in the experimental result in terms of F-score and AUC (Area Under The Curve) measure, differential privacy in addition to the encryption decreases the performances of the classifier, this imply that the usage of really large datasets is needed in order to mitigate the effect of noise introduced by differential privacy.
One of the most important works that is considered to be the first and unique example of application of HE to CNNs is the one presented by Dowlin et al. (2016) [15]. The difference with respect all the other works is given by the fact that CNNs are really powerful classification machine learning model and are way more complex than ELM or logistic regression. They propose a method to convert learned neural networks to CryptoNets, neural networks that can be applied to encrypted data. This work focuses only on the inference stage and the assumption is that the cloud (that has to perform the prediction) already has the model. CryptoNets are able to make predictions on encrypted data coming from a data provider on an outsourced server that provides back to the data owner the results of computation, still in an encrypted form. The data owner then is able to decrypt the results and access to the unencrypted prediction. In this sense the purpose of this work is similar to the one of this thesis, however they do not provide an open source library for CNNs that works homomorphically on encrypted data that can reproduce their work and they do not focus on the estimate of optimal parameters for the HE conversion of the CNN, even if this is an important starting point to achieve better performances. An other difference is the fact that they apply batching techniques in order to pack more input data in the same ciphertext, using the Chinese Reminder Theorem (CRT) [19] to perform Single Instruction Multiple Data (SIMD) operations [23].This gives good throughput but relatively poor latency in terms of predictions per hour.

2.3 Other related works

One of the first attempt made to speed up the computation proposed in CryptoNets [15], as presented in the previous Section 2.2, has been made by Yizhi Wang et al. (2018) in [42]. In particular they propose an hardware solution, by providing a dedicated convolution core architecture for the most complex convolutional layers of CryptoNets on the basis of Fan-Vercauteren (FV) HE scheme [20]. They propose a simplified modular multiplication algorithm suitable for CryptoNets that enable to simplify and reduce dramatically hardware complexity. Compared to a well optimized version CPU implementation of CryptoNets, their proposed architecture is $11.9\times$ faster while consuming a power of $537mW$.
Zhenyong Zhang et al. (2018) in [44] propose the application of HE for secure Kalman Filter state estimation in cyber-physical systems. In their scenario, the data providers are multiple sensors that send their measurements to an outsourced estimator node in charge of computing the Kalman Filter prediction. The goal of using HE is to protect data against confidentiality attacks in

the communication network or at the estimator node. They consider that sensor measurements are valid but they can be overheard during communications. Furthermore, they assume a curious- but-honest estimator; i.e., the estimator carries out the estimation faithfully, but it may try to learn as much as possible about the inputs and/or outputs of the estimation. They use RSA since there is only the need of a multiplicative homomorphic encryption.

Chapter 3

Background

3.1 Homomorphic Encryption

Traditional encryption schemes, both symmetric and asymmetric, were not designed to respect any algebraic structure of the plaintext and ciphertext spaces, i.e. no computations can be performed on the ciphertext in a way that would pass through the encryption to the underlying plaintext without using the secret key, and such a property would in many contexts be considered a vulnerability. An encryption scheme that allows computations to be done directly on the encrypted data is said to be a *homomorphic encryption scheme* [30]. In mathematics, a homomorphism is a *structure-preserving* transformation. For example, consider the map $\Phi : \mathbb{Z} \longrightarrow \mathbb{Z}_7$ such that $\Phi(z) := z \ (mod\ 7)$. This map Φ preserves both the additive and the multiplicative structure of the integers in the sense that for every $z_1, z_2 \in \mathbb{Z}$ we have that $\Phi(z_1 + z_2) = \Phi(z_1) \oplus \Phi(z_2)$ and $\Phi(z_1 \cdot z_2) = \Phi(z_1) \otimes \Phi(z_2)$ where $\oplus$ and $\otimes$ are the addition and multiplication operations in $\mathbb{Z}_7$. The map Φ is a ring homomorphism (see Subsection 3.2.1) between the rings $\mathbb{Z}$ and $\mathbb{Z}_7$ [15]. Finding a general method for computing on encrypted data had been a goal in cryptography since it was proposed in 1978 by Rivest, Adleman and Dertouzos [38]. Since there the interest in this topic is grown due to its numerous applications. Craig Gentry [22] was the first to propose a plausible construction for a fully homomorphic scheme that respects both addition and multiplication and since then new and more efficient schemes has been proposed, but despite the promising theoretical power of homomorphic encryption, the practical side remained underdeveloped for a long time.

3.1.1 Classification of Homomorphic Encryption Schemes

It is worth to analyze different classes of homomorphic schemes to clarify the potentiality of this type of encryption as described in [4].
Given that a circuit C is a series of computations made on some inputs, it is possible to state some preliminary definitions:

Definition 2. ($\mathcal{C}$-Evaluation Scheme). Let $\mathcal{C}$ be a set of circuits. A $\mathcal{C}$-evaluation scheme for C is a tuple of probabilistic polynomial-time algorithms (Gen, Enc, Eval, Dec) such that:

$\mathsf{Gen}(1^\lambda, \alpha)$ is the key generation algorithm. It takes two inputs, security parameter λ and auxiliary input α, and outputs a key tuple (pk, sk, evk), where pk is the public key used for encryption, sk is the secret key used foe decryption and evk is the key used for the Eval algorithm.

$\mathsf{Enc}(pk, m)$ is the encryption algorithm. As input takes the encryption key pk and a plaintext m. It outputs a ciphertext c.

$\mathsf{Eval}(evk, C, (c_1, ..., c_n))$ is the evaluation algorithm. It takes as input the evaluation key evk[1], a circuit $C \in \mathcal{C}$ and a tuple of inputs $(c_1, ..., c_n)$ that can be a mix of ciphertexts and previous evaluation results. It produces an evaluation output.

$\mathsf{Dec}(sk, c)$ is the decryption algorithm. It takes as input the decryption key sk and either a ciphertext or an evaluation output and produces a plaintext m.

Definition 3. (Correct Decryption). A $\mathcal{C}$-evaluation scheme (Gen, Enc, Eval, Dec) is said to *correctly decrypt* if for all m in the plaintext space,

$$\mathsf{Pr}[\mathsf{Dec}(sk,\mathsf{Enc}(pk, m) = \mathsf{m}] = 1$$

This means that it must be able to decrypt a ciphertext to the correct plaintext without error.

Definition 4. (Correct Evaluation [10]) A $\mathcal{C}$ - evaluation scheme (Gen, Enc, Eval, Dec) *correctly evaluates* all circuits in $\mathcal{C}$ if for all inputs c_i, with $i = 1, ..., n$, for every $C \in \mathcal{C}$,

$$\mathsf{Pr}[\mathsf{Dec}(sk, \mathsf{Eval}(evk, C, c_1, ..., c_n)) = C(m_1, \ldots, m_n)] = 1 - \epsilon(\lambda)$$

where $m_i \leftarrow \mathsf{Dec}(sk, c_i)$ and ϵ is a negligible function.
This means that with overwhelming probability, decryption of the homomorphic evaluation of a permitted circuit yields the correct result.
A $\mathcal{C}$-evaluation scheme is *correct* if it satisfies the properties of correct evaluation and correct decryption. It is also *compact* if the ciphertext size does not grow too much through homomorphic operations and the output length only depends on the security parameter λ.
The classification of schemes is based on which kind of circuits the scheme itself can evaluate.

[1]The evaluation key is used to perform the relinearization operation as explained in Subsection 3.2.4

Somewhat Homomorphic. A $\mathcal{C}$-evaluation scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ with correctness property is called *somewhat homomorphic* encryption scheme (SHE).
There are no guarantees for compactness and the set $\mathcal{C}$ of permitted circuits does not contain all the circuits.

Leveled Homomorphic [10]. A $\mathcal{C}$-evaluation scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ is called a *leveled homomorphic scheme* if it takes an auxiliary input $\alpha = d$ to Gen which specifies the maximum depth of circuits that can be evaluated. Its properties are correctness and compactness. The latter implies that the *length of the evaluation output* does not depend on d.

The difference between SHE and leveled HE is that the depth of circuits which a SHE can handle can be increased through parameters choice - this usually means that the ciphertext size increases with the depth of the circuit allowed, while for a leveled HE the maximum depth is an input parameter and the length of the ciphertext does not depend on it.

Fully Homomorphic Encryption [10]. A $\mathcal{C}$-evaluation scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ that is compact, correct, and where $\mathcal{C}$ is the set of all circuits is called *fully homomorphic* encryption scheme.
This definition means that the scheme can evaluate any circuit of arbitrary size, which does not need to be known when setting the parameters.
However it is fair to say that FHE mostly exists on papers.

3.1.2 Limitations of Homomorphic Encryption

Despite HE is a powerful instrument with many possible applications in real-world scenarios, it has some limitations that must be taken into account.

Support for multiple users. Traditional FHE schemes are *single-key* in the sense that they can perform (arbitrarily complex) computations on inputs encrypted under the same key. Nevertheless, Gentry [21] proposed a way of using single key FHE scheme in order to do multiparty computation, by using a joint public key and a *shared-secret key* among all parties. However it requires an *interactive* cooperations of all parties during computations. López-Alt et al. [31] have shown promising direction to address this problem by proposing an *N-key fully homomorphic encryption scheme.* Despite these results, at the time of writing, *multi-key* FHE are not very efficient and there are no practical implementations in any library.

Large computational overhead. It describes the ratio between the computation time in the encrypted version versus computation time in the clear. Although polynomial in size, this overhead tends to be a rather large polynomial, which increases runtimes substantially and makes homomorphic computation of complex functions impractical. Nevertheless it is not easy

to quantify this overhead since is platform dependant and can differ also depending on the particular HE scheme chosen and its practical implementation. One benchmark commonly used is the homomorphic evaluation of the AES [13] circuit, since it is nontrivial but also not completely infeasible. One study [23] based on the RLWE Brakerski-Gentry-Vaikuntanathan cryptosystem [9] and on its library implementation HElib [27], shows that it takes about 4 minutes and 3GB of RAM, running on a laptop Intel Core i5-3320M running at 2.6GHz, to evaluate an entire AES-128 encryption operation. Even if by using Single Instruction Multiple Data techniques, it is possible to process up to 120 blocks in each such evaluation, yielding an amortized rate of just over 2 seconds per block. This implies that performances can be similar to the non-encrypted version of the AES circuit, but at the cost of increased memory consumption.

Some inefficient operations. For example there is no efficient way to implement the comparison operation, because it requires dealing with binary messages [33]. This latter representation brings an important issue: to perform an integer addition or multiplication with the binary representation, one must reconstruct the binary circuit of the operators.

FHE does not necessarily imply secret function evaluation. Indeed its goal is to allow the evaluation of arithmetic circuits on encrypted inputs, without revealing the input wire values to the evaluator. In particular, no attempt is made to keep any information hidden from the owner of the secret key [30]. For example, the homomorphic execution of the forward phase of a CNN on encrypted data does not guarantee the secrecy of the CNN model (i.e. anyone could see the structure of the network). However it is possible to solve this problem and obtain *function privacy* in a number of ways. One way already described by Gentry in [22] is to flood additional noise in the ciphertext. An other way is to use the GSW cryptosystem [24] as described in [8].

Malleability. It is a property of some cryptographic algorithms [14]. It is the ability to transform a ciphertext into a different ciphertext which will produce a new and different plaintext when decoded. That is, given an encryption of a plaintext m, it is possible to generate another ciphertext which decrypts to $f(m)$, for a known function f, without necessarily knowing or learning m. For example, suppose that a user sends an encrypted message o a bank containing, say, "TRANSFER $0000100.00 TO ACCOUNT #199." If an attacker can modify the message on the wire, and can guess the format of the unencrypted message, the attacker could be able to change the amount of the transaction, or the recipient of the funds, e.g. "TRANSFER $0100000.00 TO ACCOUNT #227". However, homomorphic encryption systems are malleable by design, allowing authorized parties to alter enciphered text without knowing the contents of the message. That is, they can manipulate an encrypted stream (set a flag, add a value) without having the ability to decipher the stream, or without

expending the effort to decrypt and re-encrypt the stream. Moreover if an encrypted message is sent to an external server for outsourced computation, it must be considered a trusted party. Indeed no one can guarantee that the server has performed the desired computation. Even the RSA is a malleable cryptosystem, however if used together with padding methods such as OAEP [7] this issue can be solved.

3.2 Brakerski/Fan-Vercauteren scheme (BFV)

One of the practical implementations of Gentry scheme [22] is the FHE conjectured by Brakerski, Fan and Vercauteren (BFV)[20]. In this section this homomorphic scheme will be described in details since it is the one used for the practical part of this thesis.

3.2.1 Ring Definition

The BFV relies on the concept of ring, in particular on the commutative rings [19].
A commutative ring R is a set on which there are two operations defined: addition and multiplication, such that $0 \in R$ is the identity element of addition and $1 \in R$ is the identity of the multiplication operation. For every element $a \in R$ there exists an element $-a \in R$ such that $a + (-a) = 0$. Furthermore, for every $a, b, c \in R$:

$a(bc) = (ab)c$; (associativity of product)
$a(b + c) = ab + ac$; (left associativity of product w.r.t addition)
$(a + b)c = ac + bc$; (right associativity of product w.r.t. addition)
$a + (b + c) = (a + b) + c$; (associativity of addition)
$a + b = b + a$; (commutative addition)
$ab = ba$. (commutative product)

The set of integers $\mathbb{Z}$ is a ring as as well as the set $\mathbb{Z}_m$ of integers modulo m. If R is a given commutative ring, then the set of all polynomials in the variable x whose coefficients are in R forms the polynomial ring, denoted $R[x]$. In this work the focus will be on the ring $\mathbb{Z}_m[x]$ of polynomials with integer coefficients modulo m and in particular on the polynomial ring $\mathbb{Z}_m[x]/(x^n + 1)$, that is polynomials of degree less than n with coefficients modulo m.

3.2.2 Basic Notation

- $R = \mathbb{Z}[x]/(f(x))$ is the polynomial ring where $f(x) \in \mathbb{Z}[x]$ is a monic irreducible polynomial of degree n. In practice the cyclotomic polynomial $\Phi_m(x)$, i.e. the minimal polynomial of the primitive m-th roots of unity, is used. The most popular choice for expository purposes is to take $f(x) = x^n + 1$ with $n = 2^d$.

- Elements of the ring R will be denoted in lowercase bold, e.g. $\mathbf{a}\in R$. The coefficients of an element $\mathbf{a}\in R$ will be denoted by a_i, i.e. $\mathbf{a} = \sum_{i=0}^{n-1} a_i \cdot x^i$.
- $\| \mathbf{a} \| = max_i \mid a_i \mid$ is the infinity norm.
- $\delta_R = max \{||\mathbf{a}\cdot\mathbf{b}||/(||\mathbf{a}||\cdot||\mathbf{b}||) : \mathbf{a}, \mathbf{b}\in R\}$ is the expansion factor of R.
- Let $q > 1$ be an integer, then $\mathbb{Z}_q$ denotes the *set* of integers $(-q/2, q/2]$.
- R_q is the set of polynomials in R with coefficients in $\mathbb{Z}_q$.
- For each $a\in\mathbb{Z}$, is denoted by $[a]_q$ the unique integer in $\mathbb{Z}_q$ such that $[a]_q = a \mod q$.
- The reduction to the interval $[0, q)$ will be denoted as $r_q(a)$ (reminder modulo q).
- Given a probability distribution $\mathcal{D}$, $x \longleftarrow \mathcal{D}$ denotes that x is sampled from $\mathcal{D}$, while for a set S, $x \longleftarrow$ S denotes that x is sampled uniformly from S.
- The discrete Gaussian distribution $D_{\mathbb{Z},\sigma}$ over the integers is the probability distribution that assigns a probability proportional to $exp(-\pi|x|^2/\sigma^2)$ to each $x\in\mathbb{Z}$. It is used to define a distribution χ on R.
- σ is the standard deviation of the discrete Gaussian distribution, with default value of $3.19 \approx 8/2\pi$.
- B bound on the distribution χ, that is it is supported on $[-B, B]$.

3.2.3 RLWE Problem

Most of HE schemas base their security on the hardness of the Ring-Learning With Errors problem. The RLWE problem is a ring based version of the LWE problem. In [37] Regev gave a quantum reduction of certain approximate *Shortest Vector Problem* (SVP) to LWE, i.e. if one can solve LWE, then there is a quantum algorithm to solve certain approximate SVP. Informally, the SVP requires a player to provide a shortest possible vector in a given lattice and it is known to be an NP-hard problem. In particular, the asymptotically fastest known algorithms for obtaining an approximation to SVP on ideal lattices to within polynomial factors require time $2^{\Omega(n)}$ [32]. Thus, the only evidence supporting the conjecture that LWE is as hard as SVP is the fact that there are no known quantum algorithms for lattice problems that outperform classical algorithms, even though this is probably one of the most important open questions in the field of quantum computing.

Definition 5. (**Decision-RLWE**) For a security parameter λ, let $f(x)$ be a cyclotomic polynomial $\Phi_m(x)$ with deg(f) = φ(m) depending on λ and $R = \mathbb{Z}[x]/(f(x))$. Let $q = q(\lambda) \geq 2$ be an integer. For a random element $\mathbf{s} \in R_q$ and a distribution $\chi = \chi(\lambda)$ over R, denote with $A^{(q)}_{\mathbf{s},\chi}$ the distribution obtained by

choosing a uniformly random element $\mathbf{a} \longleftarrow R_q$ and a noise term $\mathbf{e} \longleftarrow \chi$ and outputting $(\mathbf{a}, [\mathbf{a}\cdot\mathbf{s} + \mathbf{e}]_q)$. The decision-RLWE problem is to distinguish $A^{(q)}_{\mathbf{s},\chi}$ and the uniform distribution $U(R_q^2)$.

3.2.4 Encryption Scheme

From the decision problem reported in Subsection 3.2.3 it is possible to derive the following encryption scheme originally described in the extended version of [32].
The plaintext space is taken as R_t for some integer $t > 1$. Let $\Delta = \lfloor q/t \rfloor$ and denote with $r_t(q) = q \; mod \; t$, it follows that

$$q = \Delta \cdot t + r_t(q), \tag{3.1}$$

with q and t that do not have to be prime nor coprime. The definition of the scheme mostly follows the one given in [20].

$\mathsf{SH.SecretKeyGen}(1^\lambda)$ sample $\mathbf{s} \leftarrow \chi$ and output $sk = \boldsymbol{s}$

$\mathsf{SH.PublicKeyGen}(\mathsf{sk})$ set $\boldsymbol{s} = sk$, sample $\boldsymbol{a} \leftarrow R_q$, small error $\boldsymbol{e} \leftarrow \chi$ and output

$$pk = (\boldsymbol{p}_0, \boldsymbol{p}_1) \coloneqq ([-(\boldsymbol{a}\cdot\boldsymbol{s} + \boldsymbol{e})]_q, \boldsymbol{a})$$

$\mathsf{SH.EvalateKeyGen}(\mathsf{sk}, \mathsf{T})$ for $i = 0, ..., l = \lfloor logT(q) \rfloor$ and T is a base (independent of t), sample $\boldsymbol{a}_i \leftarrow R_q$, $\boldsymbol{e}_i \leftarrow \chi$ and output

$$rlk = \left[\left([-(\boldsymbol{a}_i\cdot\boldsymbol{s} + \boldsymbol{e}_i) + T^i \cdot \boldsymbol{s}^2]_q, \boldsymbol{a}_i \right) : i \in [0..l] \right]$$

The *evaluation keys* are used in the relinearization phase which goal is to decrease the size of the ciphertext back to (at least) 2 after it has been increased by multiplications.

$\mathsf{SH.Encrypt}(\mathsf{pk}, \boldsymbol{m})$ to encrypt a message $\boldsymbol{m} \in R_t$, sample $\boldsymbol{u}, \boldsymbol{e}_1, \boldsymbol{e}_2 \leftarrow \chi$ and output

$$ct = (\boldsymbol{c}_0, \boldsymbol{c}_1) \coloneqq ([\boldsymbol{p}_0\cdot\boldsymbol{u} + \boldsymbol{e}_1 + \Delta\cdot\boldsymbol{m}]_q, [\boldsymbol{p}_1\cdot\boldsymbol{u} + \boldsymbol{e_2}]_q)$$

$\mathsf{SH.Decrypt}(\mathsf{sk}, \mathsf{ct})$ set $\boldsymbol{s} = sk$ and $ct = (\boldsymbol{c}_0, \boldsymbol{c}_1)$ and compute

$$\left[\left\lfloor \frac{t \cdot [\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}]_q}{q} \right\rfloor \right]_t$$

3.2.5 Noise

All existing schemes have the common trait that they add a small "noise" component during encryption. Computing homomorphically on ciphertexts will cause these noises to grow up to the point when they become so large that decryption fails. In particular the noise growth caused by homomorphic multiplication has been the major obstacle to designing efficient schemes.[20]

Definition 6. (Invariant Noise)[30] Let $\mathsf{ct} = (c_0, c_1)$ be a ciphertext encrypting the message $m \in R_t$. Its invariant noise v is the polynomial with the smallest infinity norm such that

$$\frac{t}{q}\mathsf{ct}(s) = \frac{t}{q}(c_0 + c_1 s) = m + v + at \tag{3.2}$$

for some polynomial a with integer coefficients.

Lemma 7. [30] *The function* $\mathsf{SH.Decrypt}(\mathsf{sk}, \mathsf{ct})$ *as presented in Subsection 3.2.4 correctly decrypts the ciphertext* ct *encrypting a message m, as long as the invariant noise v satisfies* $||v|| < 1/2$.
Proof. Let $\mathsf{ct} = (c_0, c_1)$. Using the formula for decryption, for some polynomial A with integer coefficients:

$$\begin{aligned} m' &= \left[\left\lfloor \frac{t}{q}[c_0 + c_1 \cdot s]_q \right\rceil\right]_t \\ &= \left[\left\lfloor \frac{t}{q}(c + c_1 \cdot s + Aq) \right\rceil\right]_t \\ &= \left[\left\lfloor \frac{t}{q}(c + c_1 \cdot s) + At \right\rceil\right]_t \\ &= \left[\left\lfloor \frac{t}{q}(c + c_1 \cdot s) \right\rceil\right]_t. \end{aligned}$$

Then by definition of invariant noise in Eq.3.2,

$$m' = [\lfloor m + v + at \rceil]_t = m + \lfloor v \rceil.$$

Hence decryption is successful as long as v is removed by the rounding, i.e. if $||v|| < 1/2$.

Definition 8. (Noise Budget)[30] Let v be the invariant noise of a ciphertext ct encrypting a message $m \in R_t$. Then the noise budget of ct is $-log_2(2||v||)$.

In other words, the *(Invariant) Noise Budget (NB)* is the left noise in the ciphertext before decryption fails.

Lemma 9. [30] *The function* $\mathsf{SH.Decrypt}(\mathsf{sk}, \mathsf{ct})$ *as presented in Subsection 3.2.4 correctly decrypts the ciphertext* ct *encrypting a message m, as long as the noise budget of* ct *is positive.*

Even a freshly encrypted ciphertext has a non zero amount of noise.

Lemma 10. *(Initial Noise)*[30] *Let* $\mathsf{ct} = (c_0, c_1)$ *be a fresh encryption of a message* $m \in R_t$*. The noise v in* ct *satisfies*

$$||v|| \leq \frac{r_t(q)}{q}||m|| + \frac{tB}{q}(2n+1).$$

3.2.6 Operations and Noise Growth

Given the interpretation of the ciphertext $ct(\boldsymbol{s})$ as in Eq.3.2 it is possible to derive homomorphic addition $\mathsf{SH.ADD}$ and multiplication $\mathsf{SH.MUL}$.

Addition

$\mathsf{SH.ADD}(\mathsf{ct_1}, \mathsf{ct_2}) \coloneqq ([\mathsf{ct_1}[0] + \mathsf{ct_2}[0]]_\mathsf{q}, [\mathsf{ct_1}[1] + \mathsf{ct_2}[1]]_\mathsf{q})$.

It is possible to show that if $\mathsf{ct_1}$ and $\mathsf{ct_2}$ have noise v_1 and v_2 respectively, then the noise v_{add} in their sum is $v_{add} = v_1 + v_2$ and satisfies $||v_{add}|| \leq ||v_1|| + ||v_2||$.

Multiplication

$\mathsf{SH.MUL}(\mathsf{ct_1}, \mathsf{ct_2}, \mathsf{rlk})$: compute

$$\boldsymbol{c}_0 = \left[\left\lfloor \frac{t \cdot (ct_1[0] + \cdot ct_2[0])}{q} \right\rceil\right]_q$$
$$\boldsymbol{c}_1 = \left[\left\lfloor \frac{t \cdot (ct_1[0] \cdot ct_2[1] + ct_1[1] \cdot ct_2[0])}{q} \right\rceil\right]_q$$
$$\boldsymbol{c}_2 = \left[\left\lfloor \frac{t \cdot (ct_1[1] + \cdot ct_2[1])}{q} \right\rceil\right]_q$$

The multiplication makes grow the ciphertext size, for this reason a relinearization step is necessary in order to reduce it. As showed in $\mathsf{SH.RELIN}$ operation. It is possible to prove that: if $\mathsf{ct_1} = (x_0, \ldots, x_{j_1})$ is a ciphertext of size $j_1 + 1$ encrypting m_1 with noise v_1, $\mathsf{ct_2} = (y0, \ldots, y_{j_2})$ is a ciphertext of size $j_2 + 1$ encrypting m_2 with noise v_2, N_{m_1} and N_{m_2} be upper bounds on the number of non-zero terms in the polynomials m_1 and m_2 respectively, then the noise v_{mult}

in the product $\mathsf{ct}_{\mathsf{mult}}$ satisfies the following bound

$$||v_{mult}|| \leq \left[(N_{m_1}+n)||m_1|| + \frac{nt}{2}\frac{n^{j_1+1}-1}{n-1})\right]||v_2|| + \left[(N_{m_2}+n)||m_2|| + \frac{nt}{2}\frac{n^{j_2+1}-1}{n-1})\right]||v_1|| + 3n||v_1||||v_2|| + \frac{t}{2q}\frac{n^{j_1+j_2+1}-1}{n-1}).$$

Relinearization
$\mathsf{SH.RELIN}(\mathsf{ct}_{\mathsf{mult}}, \mathsf{rlk})$: write $\boldsymbol{c}_2$ in base T, i.e. write $\boldsymbol{c}_2 = \sum_{i=0}^{l} \boldsymbol{c}_2^{(i)} T^i$ with $\boldsymbol{c}_2^{(i)} \in R_T$ and set:

$$\boldsymbol{c}_0' = \left[\boldsymbol{c}_0 + \sum_{i=0}^{l} rlk[i][0] \cdot \boldsymbol{c}_2^{(i)}\right]_q$$
$$\boldsymbol{c}_1' = \left[\boldsymbol{c}_1 + \sum_{i=0}^{l} rlk[i][1] \cdot \boldsymbol{c}_2^{(i)}\right]_q$$

output $\mathsf{ct}_{\mathsf{relin}} = (\boldsymbol{c}_0', \boldsymbol{c}_1')$.
It is possible to show that if $\mathsf{ct}_{\mathsf{mult}}$ is a ciphertext of size *M+1* encrypting m, and having noise v, and $\mathsf{ct}_{\mathsf{relin}}$ of size *N+1* is the ciphertext encrypting m, obtained by the relinarization of $\mathsf{ct}_{\mathsf{mult}}$ where $2 \leq N+1 \leq M+1$, then the noise v_{relin} in $\mathsf{ct}_{\mathsf{relin}}$ can be bounded as $||v_{relin}|| \leq ||v|| + \frac{t}{q}(M-N)nB(l+1)T$.

There are two other important operations $\mathsf{AddPlain}(ct, m_{add})$and $\mathsf{MultiplyPlain}(\mathsf{ct}, \mathsf{m}_{\mathsf{mul}})$ that can be seen as a particular case of the $\mathsf{SH.ADD}(\mathsf{ct}_1, \mathsf{ct}_2)$, $\mathsf{SH.MUL}(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{rlk})$. They can hugely improve performance of addition and multiplication operations. They are useful in the case the operands are a given ciphertext ct encrypting a plaintext polynomial m and an unencrypted plaintext polynomial m_{add} or m_{mul} that does not need to be protected. It is possible to show mathematically how this is possible and which are the advantages in terms of grow of noise budget that these operations can bring with respect to SH.ADD and SH.MUL.

Plain Multiplication

Lemma 11. [30]*(*MULTIPLY PLAIN*) Let* $\mathsf{ct} = (x_0, x_1)$ *be a ciphertext encrypting* m_1 *with noise* v*, and let* m_2 *be a plaintext polynomial. Let* N_{m_2} *be an upper bound on the number of non-zero terms in the polynomial* m_2*. Let* $\mathsf{ct}_{\mathsf{pmult}}$ *denote the ciphertext obtained by plain multiplication of* ct *with* m_2*. Then the noise in the plain product* $\mathsf{ct}_{\mathsf{pmult}}$ *is* $v_{pmult} = m_2 v$*, and can be bounded as*

$$||v_{pmult}|| \leq N_{m_2}||m_2||\,||v||.$$

Proof. By definition $ct_{pmult} = (m_2x_0, m_2x_1)$. Hence for some polynomials a, a' with integer coefficients,

$$\begin{aligned}\frac{t}{q}\mathsf{ct}_{\mathsf{pmult}}(s) &= \frac{t}{q}(m_2x_0 + m_2x_1s)\\ &= m_2\frac{t}{q}(x_0 + x_1s)\\ &= m_2\frac{t}{q}ct(s)\\ &= m_2(m_1 + v + at) \textit{ (see Eq. 3.2)}\\ &= m_1m_2 + m_2v + m_2at\\ &= [m_1m_2]_t + m_2v + (m_2a - a')t,\end{aligned}$$

where in the last line has been used $[m_1m_2]_t = m_1m_2 + a't$. Hence the noise is $v_{pmult} = m_2v$ and can be bounded as

$$||v_{pmult}|| \leq N_{m_2}||m_2||\,||v||$$

Plain Addition

Lemma 12. [30]*(ADD PLAIN) Let* $\mathsf{ct} = (x_0, x_1)$ *be a ciphertext encrypting* m_1 *with noise* v*, and let* m_2 *be a plaintext polynomial. Let* $\mathsf{ct}_{\mathsf{padd}}$ *denote the ciphertext obtained by plain addition of* ct *with* m_2*. Then the noise in* $\mathsf{ct}_{\mathsf{padd}}$ *is* $v_{padd} = v - \frac{r_t(q)}{q}m_2$*, and the bound is*

$$||v_{padd}|| \leq ||v|| + \frac{r_t(q)}{q}||m_2||.$$

Proof. By definition of plain addition $\mathsf{ct}_{\mathsf{padd}} = (x_0 + \Delta m_2, x_1)$. Hence for some polynomials a, a' with integer coefficients,

$$\begin{aligned}\frac{t}{q}\mathsf{ct}_{\mathsf{padd}}(s) &= \frac{t}{q}(x_0 + \Delta m_2 + x_1s)\\ &= \frac{\Delta t}{q}m_2 + \frac{t}{q}(x_0 + x_1s)\\ &= \frac{\Delta t}{q}m_2 + \frac{t}{q}ct(s)\\ &= m_1 + v + \frac{q - r_t(q)}{q}m_2 + at \textit{ (see Eq.3.1)}\\ &= m_1 + m_2 + v - \frac{r_t(q)}{q}m_2 + at\\ &= [m_1 + m_2]_t + v - \frac{r_t(q)}{q}m_2 + (a - a')t,\end{aligned}$$

where in the last line has been used $[m_1 + m_2]_t = m_1 + m_2 + a't$. Hence the noise is $v_{padd} = v - \frac{r_t(q)}{q} m_2$ and can be bounded as

$$||v_{padd}|| \leq ||v|| + \frac{r_t(q)}{q}||m_2||.$$

3.2.7 Encoding

In Subsection 3.2.4 is shown that plaintext elements in the BFV scheme are polynomials in R_t and homomorphic operations on ciphertexts are reflected in the plaintext side as corresponding operations in the ring R_t. Thus it is important to find a mapping between integers (or real numbers) and polynomials in R_t in an appropriate way in order to make applicable HE in the real world. Since no non-trivial subset of $\mathbb{Z}$ is closed under additions and multiplications, it is needed to settle for something that does not respect an arbitrary number of homomorphic operations. It is then the responsibility of the evaluating party to be aware of the type of encoding that is used, and perform only operations such that the underlying plaintexts throughout the computation remain in the image of the encoding map [30]. In other words a number must be *encoded*.

Integer Encoding
Given an integer a it is required to choose a base S and to form the (up to n-bits) base-S expansion of $|a|$, say $(a_{n-1}x^{n-1} + \ldots + a_1x + a_0)$. When $S = 2$, $-(2^n - 1) \leq a \leq 2^n - 1$ and the expansion is a binary expansion, when $S > 2$, for the base-S expansion, the coefficients are chosen from the symmetric set $[-(S-1)/2, ..., (S-1)/2]$, since there is a unique representation with at most n coefficients for each integer in $[-(S^n - 1)/2, (S^n-1)/2]$.

$$\texttt{IntEncode}(a, S) = sign(a) \cdot (a_{n-1}x^{n-1} + \ldots + a_1x + a_0)$$

Decoding can fail for two reasons:

1. if a reduction modulo the plaintext modulus t occurs
2. if a reduction modulo the polynomial modulus $x^n + 1$ occurs

If any of these reductions happens, decoding yields an incorrect result, but no indications of these events are given until the final decryption.

Fractional Encoding
Real numbers can be encoded as integers if properly scaled and rescaled back after decryption or it is possible to encode real numbers as fixed precision numbers. Just like the integer encoding (Paragraph 3.2.7 above), the fractional encoding is parametrized by an integer base $S \geq 2$ [16]. It works by first encoding the integer part with an Integer Encoding, then the fractional part is

codified in base-S, n is added to each exponent of the fractional part and it is converted in a polynomial by changing the base B into the variable x. Next each sign of the fractional part is flipped. For any rational number r with finite base-S expansion:

$$\texttt{FractionEncode}(r, S) = sign(r) \cdot [\texttt{IntEncode}(\lfloor |r| \rfloor, S) + \texttt{FractionEncode}(\{|r|\}, S)],$$

where $\{\cdot\}$ denotes the fractional part. For example, consider $S = 2$ and $r = 5.8125$ which has finite binary expansion $5.875 = 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4}$. Its integer part is encoded as $\texttt{IntEncode}(5, S = 2) = x^2 + 1$, while its fractional part is encoded as $\texttt{FractionEncode}(0.875, S = 2) = -x^{n-1} - x^{n-2} - x^{n-4}$.
Some rational numbers have not a finite base-S expansion, thus in order to truncate them it is important to fix n_i bits for the integer part and n_f bits for the fractional part, such that $n_i + n_f \leq n$.
Decoding can fail for two reasons:

1. if any of the coefficients of the underlying plaintext polynomials wrap around the plaintext modulus t the result after decoding is likely to be incorrect, just as in the case of the integer encoding Paragraph 3.2.7

2. if during homomorphic multiplications the fractional parts of the underlying plaintext polynomials expands down towards the integer part, and the integer part expands up towards the fractional part and these different parts get mixed up.

3.3 Convolutional Neural Networks: A Brief Overview

3.3.1 Approximations For Homomorphic Encryption

Chapter 4

Parameters Estimation Methodology

4.1 Encryption Parameters

The choice of encryption parameters is crucial since it significantly affects the performance, capabilities, and security of the encryption scheme. Some choices of parameters may be insecure, give poor performance, yield ciphertexts that will not work with any homomorphic operations, or a combination of all of these. In Table 4.1 there is a list of the most important parameters of the BFV encryption scheme, that has been explained in Section 3.2 and below it is explained which are the relations that these parameters have each other.

Before to encrypt a number it must be encoded as a plaintext polynomial as explained in Subsection 3.2.7. Indeed in BFV the plaintext space is $R_t = \mathbb{Z}_t[x]/(x^n + 1)$, that is, polynomials of degree less than n with coefficients modulo t. Thus, it is important that the coefficients of the polynomials appearing throughout the computations never experience coefficients larger than the plaintext modulus t, otherwise the decryption procedure will lead to incorrect results.
The **plaintext modulus** t in the BFV scheme can be any positive integer. There is not a prescribed plaintext polynomial that is better than others, be-

Table 4.1: Notation used

Parameter	Description
q	Modulus in the ciphertext space
t	Modulus in the plaintext space
n	A power of 2

cause it depends on the calculi (i.e. in the considered case, on the structure of the CNN) and data that is needed to perform. Nevertheless, it is not easy to understand which is the maximum infinity norm of a polynomial that is reached throughout the computations, but it is a key point not to lose integrity of the expected result. The plaintext modulus influences also the so called Noise budget NB, that has been presented in Subsection 3.2.5. As explained before, the NB determines the number of operations one can perform on a ciphertext (multiplications, additions, exponentiations, both with another ciphertext or plaintext) without compromising the final result. More precisely, the value of the NB in a freshly encrypted ciphertext, given in Definition 8 can be approximated by the following formula:

$$NB \backsim log_2(\text{coeff\_modulus (q)/plain\_modulus (t)}) \text{ (bits)} \tag{4.1}$$

The polynomial $(x^n + 1)$ is the **polynomial modulus**, where $\boldsymbol{n}$ is a power of 2. This is both for security and performance reasons. Indeed as explained in Section 3.2, the discrete Gaussian distribution $D_{\mathbb{Z},\sigma}$ is used to define a distribution χ on R. The distribution χ is in general not as simple as just sampling coefficients according to $D_{\mathbb{Z},\sigma}$. However, for the polynomial (x^n+1) with $\boldsymbol{n}$ is a power of 2, χ can indeed be defined as $D^n_{\mathbb{Z},\sigma}$ [20], form which is possible to sample in a more efficient way. Increasing n significantly decreases performances since operations are performed over polynomials of bigger degree.

If the polynomial modulus is held fixed, then the choice of the **coefficient modulus** $\boldsymbol{q}$ affect: the noise budget in a freshly encrypted ciphertext (see Eq.(4.1)) and the security level (see Table 4.2). In general it is possible to take as q any integer as long as it is not too large to cause security problems. In Table 4.2 are showed the upper bounds on the bit length of q with respect to a given security level λ and a polynomial modulus degree n, assuming the standard deviation of the discrete Gaussian distribution σ to be the default value (see Section 3.2). These values are retrieved according to the most recent homomorphic encryption security standards. Table 4.2 shows that using a larger n allows for a larger q to be used without decreasing the security level, which in turn increases the noise ceiling and thus allows for larger t to be used. For this reason, decreasing q with all the other parameters held fixed, only increases the security level. On the other hand, given a certain q, choosing a big t allows to reach bigger infinity norm for polynomials during the computation, but at the same time decreases the initial NB and thus the possibility to evaluate a longer circuit (number of operations) on input data. Indeed, each operation performed adds a certain amount of noise to the initial ciphertext and the NB in a freshly encrypted number is the max amount of noise is it possible to add, as detailed in Subsection 3.2.5. Note that the NB computed is a pessimistic estimate and not a precise computed value and that the NB consumed depends not only on the type of operation performed (multiplication rather than addition), but also on the encryption parameters q and t chosen.

Table 4.2: Default pairs (n, q) for 128-bit, 192-bit, and 256-bit λ-security levels.

	Bit-length of q		
n	**128-bit security**	**192-bit security**	**256-bit security**
1024	27	19	14
2048	54	37	29
4096	109	75	58
8192	218	152	118
16384	438	300	237
32768	881	600	476

4.2 Problem Formulation

The task of preserving privacy of data provider while his data are processed by an outsourced CNN on a server can be achieved in different ways. Given that HE is used to guarantee the privacy of data, there can be two possibilities:

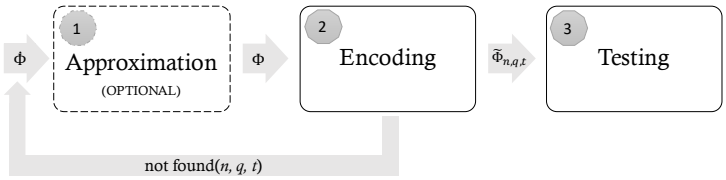
1. One can train the CNN from scratch on plain publicly available datasets or take a publicly available pre-trained CNN, transform it in a way that allows to compute only polynomial functions, to be compliant with the operations that it will be able to perform on encrypted data (as explained in Subsection 3.3.1). Then one can encode the CNN and send it to the server along with the parameters n, t and q used for the encoding. The encoded network can then be used to make prediction on encrypted data, provided that they are encrypted with the same encryption parameters.

2. The data provider can encrypt his data and send them to the server. The server trains a CNN directly on encrypted data.

The second option could look simpler than the first one, but in reality it would require to run the backward propagation algorithm homomorphically on the encrypted data and thus involve in a really computational intensive process. Moreover, since only polynomial functions could be used and thus functions like rectified linear and sigmoid functions should be omitted, some derivatives could become unbounded. This could lead to strange behavior when running the gradient descent algorithm during the backward propagation step, especially for deeper nets it would sometimes blow up or overfit.
The first option becomes the more promising way to follow and basically require the steps in Figure 4.1. At each step the accuracy of the original CNN can decrease by a factor $\epsilon \geq 0$, since it is gradually changed and HE parameters are introduced, but it becomes able to guarantee the privacy of the processed data.

STEP 1 (OPTIONAL): If the CNN $\dot{\Phi}$ in input contains non-polynomial functions, it is *approximated* with only polynomial functions as explained in Subsection 3.3.1 and the approximated CNN Φ is eventually retrained.

Figure 4.1: Transformation steps of the pre-trained CNN $\dot{\Phi}$.



STEP 2: The approximated Φ is *encoded* as explained in Subsection 3.2.7 provided that optimal encryption parameters n, t and q are found, otherwise one should return to STEP 1 to find a smaller and different CNN $\dot{\Phi}$. This step outputs $\widetilde{\Phi}_{n,q,t}$ that is an encoded network under parameters n, t and q. In Section 4.3 this step will be explained in details.

STEP 3: The encoded $\widetilde{\Phi}_{n,q,t}$ is ready to be *tested* and thus make predictions on encrypted data.

4.3 Optimization Problem

Section 4.1 gives a hint of how difficult is, in general, to find valid encryption parameters. This becomes even more difficult if the field of application of the HE is the one of Deep Learning and CNNs where there are many nested computations to perform (i. e. the depth of the circuit to compute on input data is generally high) and the number of parameters (i.e. weights) that composes the CNN's model is huge as shown in [2].
If I is a $r \times c \times d$ input image of r rows, c columns and d channels (e.g. d=3 for RGB images and d=1 for grey-scale images), $y \in \Psi = \{w_1, ..., w_\Psi\}$ is the output of the CNN representing the multi-class classification of image I and Φ is the approximated pre-trained model of a CNN, defined as $y = \Phi(I)$, there are two objectives that can be defined and can be pursued:

1. Transform the approximated CNN's pre-trained model Φ in $\tilde{\Phi}$ (*encoding*), such that the transformed $\tilde{\Phi}$ is able to accomplish the same classification task of the original Φ but on encrypted input data *Enc(I)*.

2. Find suitable encryption parameters n, t, and q that allow to perform this transformation in an optimal way.

Since homomorphic encryption supports only additions and multiplications, only polynomial functions can be computed in a straightforward way, thus the transformation of Φ in $\tilde{\Phi}$ can be achieved if Φ contains only polynomial functions. The transformation of Φ in $\tilde{\Phi}$ consists basically in an encoding of all the parameters θ of Φ, i.e., each θ becomes $\tilde{\theta}$ that is, a polynomial in the plaintext space R_t. The encoding methodology has been explained in Subsection 3.2.7. For this reason the model $\tilde{\Phi}$ is not encrypted, but only encoded, and most of the operations can be performed, in a more efficient way, as PLAIN MULTIPLICATIONS and PLAIN ADDITIONS (see Subsection 3.2.6).
More formally, the previous two objectives can be seen as the output of a function f that, given the CNN's model Φ, the plain input dataset D such that $I \in D$ with its corresponding classifications Y given by the model, the set of allowed coefficient modulus Q_{set}, the upper bound for the degree n of the polynomial modulus N_{max} , the range of values in which the plaintext modulus t can vary T_{range} and the desirable level of security λ; provides the CNN's model $\widetilde{\Phi}_{n,q,t}$ encoded with the optimal n, q and t :

$$\widetilde{\Phi}_{n,q,t} = f(\Phi, D, Y, N_{max}, Q_{set}, T_{range}, \lambda)$$

Figure 4.2 shows an approximated CNN Φ with l layers, where $\phi_{\theta_i}^{(i)}$ with $i = 1, ..., l$ is the function with parameters θ_i computed in the i-th layer, that receives as input the computation of the previous layer $i-1$. Below Φ there is $\widetilde{\Phi}_{n,q,t}$, that is the CNN transformed by the function f as explained before.

The function f can be modeled by the Optimization Problem 1. The first term of the objective function is the prediction error of the encoded CNN $\widetilde{\Phi}$, that can cause a significant loss in accuracy with respect to the one of the original model Φ. There is also the need to find the minimal n, q and t that preserve the original accuracy of the model in order to introduce the smallest possible performance's overhead in terms of memory and computational complexities required by the encoded model $\widetilde{\Phi}$. One can set parameters $\alpha, \gamma, \delta, \beta$ depending on its needs and an the application field of the CNN. For example α can be set to be small if a bigger prediction error is tolerated.
These parameters are subject to a number of constraints given by the BFV HE scheme that has been previously described in Section 3.2.
In details, as mentioned before in Section 4.1 the plaintext modulus t must be smaller than the coefficient modulus q (Eq. (4.2)) and q must be at least two (Eq. (4.3)). Constraint in Eq.(4.4) is of crucial importance as written in Section 4.1 and requires that the infinity norm of every generic computation in $\tilde{\Phi}$, both internal to a single layer or at the end of it, and thus the encrypted polynomial that represent this result, never exceed the plaintext modulus t. Constraint in Eq.(4.5) refers to the one introduced in Eq.(4.1) and states that the NB in the freshly encrypted image must be sufficiently large to support all the operations performed by each layer in $\widetilde{\Phi}$, thus the NB consumed by each computation that depends itself by the encryption parameters selected. Constraint in Eq.(4.6) is referred to the security of the encryption since the total bit count of q must be

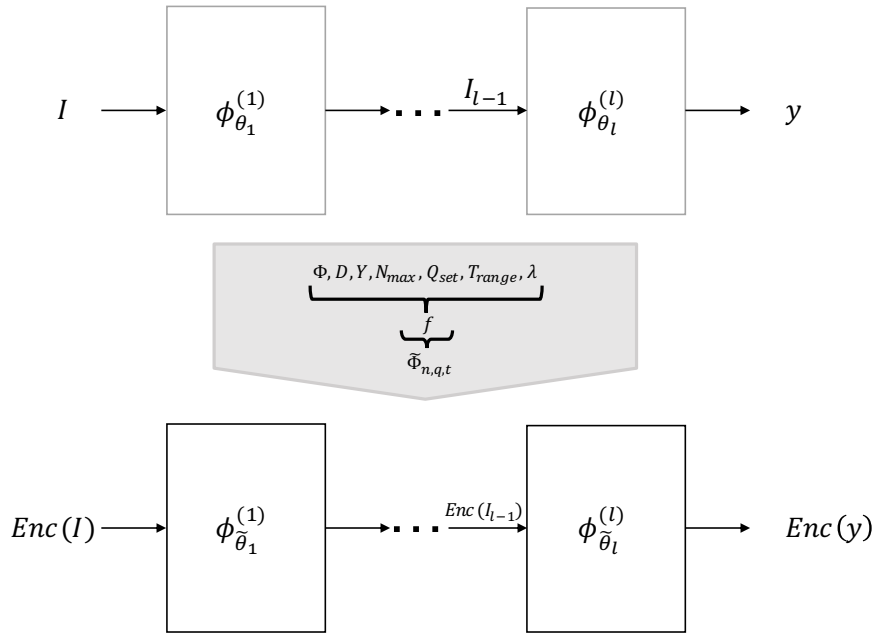


Figure 4.2: General structure of a CNN Φ and below its transformation in $\widetilde{\Phi}_{n,q,t}$ in the encryption world accomplished by function f.

Optimization problem 1

$$\min_{n,q,t} \ \alpha \sum_{i}^{D} (y_i - \tilde{y_i})^2 + \gamma t + \delta n + \beta q$$

subject to

$$t < q \tag{4.2}$$

$$q \geq 2 \tag{4.3}$$

$$t > \max ||p|| \ \text{ with } p \in R_t \text{ and generic intermediate result of } \tilde{\Phi} \tag{4.4}$$

$$\log_2 (\frac{q}{t}) > \sum_{i=1}^{l} NB_{n,q,t}(\phi^{(i)}_{\tilde{\theta}_i}) \tag{4.5}$$

$$log_2 q \leq UB(\lambda, n) \tag{4.6}$$

$$n = 2^d \ s.t. \ d \in N \tag{4.7}$$

$$t, q, n \in \mathbb{N}$$

$$\alpha, \beta, \gamma, \delta \geq 0$$

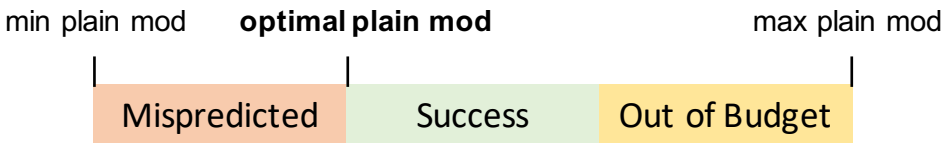
smaller than the upper bound (UB) showed in Table 4.2 for a given security level λ and polynomial modulus degree n. The last constraint (4.7) states that n must be a power of 2.
The real hard problem to solve is the exact calculus of the right term of constraint in Eq. (4.4) that is valid for every possible input data. Indeed each operation in the model $\widetilde{\Phi}$ on an input polynomial p causes a grow of the initial infinity norm of p, but since operations are performed homomorphically, i.e. the polynomial in input as first is encoded as a polynomial in R_t as described in Subsection (3.2.7) and then encrypted as a polynomial in R_q (see Subsection 3.2.4), only after decryption is achievable to understand if a reduction modulo t has occurred, since the result is different from the expected one. One can observe that is possible to estimate the grow of such infinity norm by considering the specific operation to perform (e.g. addition) and considering the shape of the two polynomials in input, before to encrypt them. However this approach can be useful only if a very small number of operations should be evaluated on the input ciphertext and this is not the case of CNNs, because otherwise this estimate would be so rough that no valid plaintext modulus t would be found, as will be informally explained in Section 5.2.
This leads to the impossibility of writing in a closed form the constraints in (4.4) and (4.5) and suggests the need to find an alternative solution for the problem that function f should solve.

4.4 Heuristic: Binary Search Of Plaintext Modulus

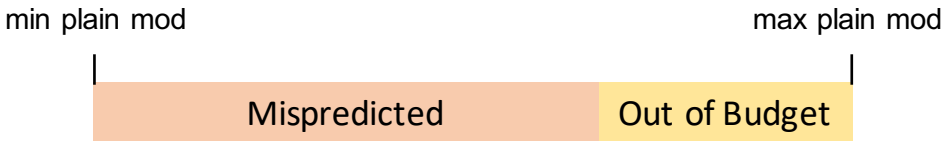
The idea is to implement a heuristic method able to find a suboptimal plaintext modulus $\tilde{t}$, that approximates the optimal t with a certain error ϵ.
The heuristic is a sort of binary search that gets as inputs: the model Φ, a number of images to test $|D|$, an ordered list t_{list} of possible plaintext moduli t with two indexes pointing to the minimum and maximum value of that list, a security parameter λ and a tentative staring big coefficient modulus q_0. It recursively testes as t the number in the middle of the two boundaries. By testing, is intended the execution of the forward phase on $|D|$ images sampled uniformly from the dataset D as showed by the pseudocode 4.2 on $\widetilde{\Phi}_{n,q,t}$. Let $\tilde{y_i} = Dec(\tilde{\Phi}(Enc(I)) = Dec(Enc(y_i))$, if the testing for a given t is successful, i.e. for all tested images i $\tilde{y_i} = y_i$, (that is the prediction for image i is the same given by the original CNN for the same i) or if the starting NB in the ciphertext is not sufficient to carry on the entire computation util the final layer, the search continues on the left interval, to find if it exists, a smaller plaintext modulus. Instead, if exists an image i such that $\tilde{y_i} \neq y_i$, i. e. there is a misprediction with respect to the original CNN Φ, the search continues on the right interval. Note that if the search is performed only on the powers of two, i.e. $t_{list} = [2, 2^2, 2^3.., 2^{59}]$, for the formula of the NB 4.1, the initial amount of NB changes significantly between different runs of the binary search, while

if the search is carried on all integers i.e. $t_{list} = [2, 3, 4.., 2^{59}]$ the difference in NB between different runs of the algorithm is not really significant. Algorithm 4.1 shows the pseudocode of the described binary search, while Algorithm 4.2 describe in details how the testing is performed.

The Algorithm 4.1 is able to find a parameter $\tilde{t}$ for every given CNN in input if a subdivision as in Figure 4.3 (a) is possible for the numbers in the given range. The found $\tilde{t}$ lies in the SUCCESS interval and tends to be as close as possible to the optimal t. The case of Figure 4.3 (b) instead, shows the worst scenario where is not found any valid $\tilde{t}$. The only possible solution is to increase the stating q_0, so to reduce the OUT OF BUDGET INTERVAL and run again the binary search, to bring back to case (a).



(a) Optimal case



(b) Worst case

Figure 4.3: Binary Search Cases

Algorithm 4.1 Plaintext Modulus Binary Search

Input: Model Φ, sorted list t_{list} of possible plain modulus t, five integers t_{min} and t_{max} s.t. $T_{range} = [t_{list}[t_{min}], t_{list}[t_{max}]]$, secuity level λ, given upper bound for the coefficient modulus q_0, number of images to test $|D|$

Output: Plain modulus $\tilde{t}$ or NOT_FOUND_T

function SEARCH_T($\Phi, t_{list}, t_{min}, t_{max}, \lambda, q_0, |D|$)
  if $t_{max} - t_{min} = 1$ **then** ▷ Base Case
    $test \leftarrow$ TEST_PLAIN_MOD($\Phi, t_{list}[t_{min}], \lambda, q_0, |D|$)
    if *test*=SUCCESS **then**
      return $t_{list}[t_{min}]$
    end if
    if *test*=OUT_OF_BUDGET **then**
      return NOT_FOUND_T
    end if
    if TEST_PLAIN_MOD($\Phi, t_{list}[t_{max}], \lambda, q_0, |D|$)=SUCCESS **then**
      return $t_{list}[t_{max}]$
    end if
    return NOT_FOUND_T
  end if
  $t_{index} \leftarrow t_{min} + (t_{max} - t_{min})/2$
  $test \leftarrow$ TEST_PLAIN_MOD($\Phi, t_{list}[t_{index}], \lambda, q_0, |D|$)
  if *test*=SUCCESS or *test*=OUT_OF_BUDGET **then** ▷ Go Left
    $t_{smaller} \leftarrow$ SEARCH_T($\Phi, t_{list}, t_{min}, t_{max} - 1, \lambda, q_0, |D|$)
    if $t_{smaller} > 0$ **then** ▷ smaller p_mod found
      return $t_{smaller}$
    end if
    if *test*=SUCCESS **then** ▷ smaller p_mod not found, prev ok
      return $t_{list}[t_{index}]$
    end if
    return NOT_FOUND_T ▷ smaller p_mod not found, prev not ok
  end if
  if $t_{index} \geq t_{max}$ **then** ▷ p_mod needed $\notin T_{range}$
    return NOT_FOUND_T
  end if
  return SEARCH_T($\Phi, t_{list}, t_{min} + 1, t_{max}, \lambda, q_0, |D|$) ▷ Go Right
end function

Algorithm 4.2 Test plaintext modulus

1: **global variables**
2: Y ▷ predicions given by model Φ
3: D ▷ dataset
4: **end global variables**
Input: Model Φ, four integers plain modulus to test t_{test}, secuity level λ, upper bound for the coefficient modulus q_0, number of images to test $|D|$
Output: SUCCESS, OUT_OF_BUDGET or MISPREDICTED
5: **function** TEST_PLAIN_MOD($\Phi, t_{test}, \lambda, q_0, |D|$)
6: $t \leftarrow t_{test}$ ▷ set encryption parameters
7: $q \leftarrow q_0$
8: $n \leftarrow \lambda(q_0)$
9: $sk \leftarrow$ GEN_SEC_KEY(n, q, t)
10: $pk \leftarrow$ GEN_PUB_KEY(sk)
11: $\tilde{\Phi} \leftarrow$ ENCODE(Φ, n, t) ▷ Transform each θ_i in $\tilde{\theta}_i$
12: **for** $i \leftarrow 0$ **to** $|D|$ **do**
13: $I \leftarrow$ RANDOMLY_SAMPLE_AT_UNIFORM(D)
14: $Enc(I) \leftarrow$ ENCRYPT(ENCODE$(I, n, t), pk)$
15: **try**
16: $Enc(y_i) \leftarrow \tilde{\Phi}(Enc(I)))$ ▷ Forward
17: **catch** Out_Of_Budget_Exception
18: **return** OUT_OF_BUDGET
19: **end try**
20: $\tilde{y_i} \leftarrow$ DECRYPT$(Enc(y_i), sk)$
21: **if** $\tilde{y_i} \neq y_i$ **then**
22: **return** MISPREDICTED
23: **end if**
24: **end for**
25: **return** SUCCESS
26: **end function**

Chapter 5

Practical Considerations About The Proposed Methodology

In Section 4.1the encryption parameters with their relationships has been described form a theoretical point of view depending on the underling BFV scheme that has been considered.
In this Chapter these parameters will be considered with respect to the practical implementation of the BFV scheme given by the Simple Encryption Arithmetic Library (SEAL 2.3.1).

5.1 Simple Encrypted Arithmetic Library (SEAL)

5.2 Practical Encryption Parameters

The **plaintext modulus** $\boldsymbol{t}$ in SEAL can be any positive integer at least 2 and at most 60 bits length, thus $2 \leq t \leq 2^{59}$.
In the **polynomial modulus** $(x^n + 1)$, $\boldsymbol{n}$ is a power of 2 as described before.
For performance reasons in SEAL the **coefficient modulus** $\boldsymbol{q}$ is a product of multiple small primes $q_1 \times ... \times q_k$, even if taking these q_k to be of special form does not provide any additional performance improvement. Therefore, it is possible to choose a set of arbitrary primes regarding their requirements as long as they are at most 60-bits long and $q_i = 1 \ (mod \ 2n)$ for $i \in \{1, 2, ..., k\}$. For this reason the bit length of q as referred in Table 4.2 is equal to $\sum_{i=1}^{k} log_2(q_i)$**.** These parameters are listed in the Table 5.1**.**

Table 5.1: Notation used in SEAL

Parameter	Description	Name in SEAL
q	Modulus in the ciphertext space of the form q1 × ... × qk, where qi are prime	coeff_modulus
t	Modulus in the plaintext space	plain_modulus
n	A power of 2	

5.3 Practical Optimization Problem

The Optimization Problem 1 presented in Section 4.3 must be changed according to the new and more detailed description of the parameters of Subsection 5.2.
The new formulation is given in Optimization Problem 2. These practical parameters are subject to a number of additional constraints given by the practical implementation of the BFV HE scheme [30].
In details, as mentioned before in Section 5.2 the plaintext modulus t is constrained to be an integer number between 2 and 2^{59} (Eq. (5.3)). Constraint in Eq. (5.6) refers to the practical shape of q to implement efficient modular arithmetic and to use David Harvey's algorithm for NTT as described in [28]. Constraint in Eq. (5.7) is changed with respect to the one in Eq. (4.6) according to the the redefined shape of q, which length is now given by the sum of the bits of each q_k that compose q. Constraint in Eq. (5.9) is given to have a degree n of the polynomial modulus of at least 1024, in order to have a correspondent upper bound of bits of q (see Table 4.2) that allow to have a starting NB greater than zero for at least some plaintext moduli t.
Concerning constraint in Eq. (5.4) it is possible to provide a more practical example, to give an idea of how difficult is to estimate the grow of the infinity norm by considering the specific operation to perform. In SEAL 2.3.1 there is the possibility to use an automatic parameter selection module that given a computation to perform basically builds a graph of that computation instead of actually calculating it and tries to estimates the grow of infinity norm and noise so to find appropriate overall parameters. The problem is that, for example, given an addition operation and two polynomials p_1 and p_2 to sum with infinity norm respectively a_1 and a_2 the infinity norm of $p_3 = p_1 + p_2$ is computed as $||p_3|| = a_1 + a_2$ that is a really pessimistic case, since it true only if coefficients a_1 and a_2 are associated to the same power in their respective polynomials p_1 and p_2. It is important to underline the fact that this estimate is used also to calculate the noise added to the resulting polynomial and thus that will induce a rough computation also of the right term in Eq. (5.5).

Optimization Problem 2

$$\min_{n,q,t} \ \alpha \sum_{i}^{D} (y_i - \tilde{y_i})^2 + \gamma t + \delta n + \beta q$$

subject to

$$t < q \tag{5.1}$$

$$q \geq 2 \tag{5.2}$$

$$2 \leq t \leq 2^{59} \tag{5.3}$$

$$t > \max ||p|| \ \text{ with } p \in R_t \text{ and generic intermediate result of } \tilde{\Phi} \tag{5.4}$$

$$\log_2 (\frac{q}{t}) > \sum_{i=1}^{l} NB_{n,q,t}(\phi_{\tilde{\theta}_i}^{(i)}) \tag{5.5}$$

$$q = \prod_{i=1}^{k} q_i \ s.t. \ q_i = 1 \ (mod\, 2n) \text{ and } q_i \text{ prime } \forall i = 1, ..., k \tag{5.6}$$

$$\sum_{i=1}^{k} \log_2 q_i \leq UB(\lambda, n) \tag{5.7}$$

$$n = 2^d \ s.t. \ d \in N \tag{5.8}$$

$$d \geq 10 \tag{5.9}$$

$$t, q, n \in \mathbb{N}$$

Chapter 6

CrCNN:A CNN Library Based On Homomorphic Encryption

6.1 Purpose

6.2 Structure

6.2.1 Types definition

- Ciphertext object represents a SEAL (see Section 5.1) Ciphertext
- ciphertext3D is a cube of Ciphertext objects
- floatCube is a cube of floats

6.3 Functionalities

Preconditions...
ciphertext3D encryptImage(vector < float > image, int zd, int xd, int yd)
ciphertext3D encryptAndSaveImage(vector < float > image, int zd, int xd, int yd, string file_name)
floatCube decryptImage(ciphertext3D encrypted_image)

Chapter 7

Experimental Results

This Chapter focuses on the practical conversion of a plain pre-trained CNN in a network able to make predictions on encrypted images.
The goal of this Chapter is to provide a practical example of how it is possible to implement the methodology proposed in Chapter 4, considering also the aspect introduced in Chapter 5. In particular, the metamorphosis of a starting plain CNN along all the steps described in Section 4.2 and reported in Figure 4.1 will be followed and the loss of accuracy caused by this procedure will be tracked.
The heuristic proposed in Section 4.4 will be tested on the given CNN and the algorithm will be examined considering its speed to converge to a reliable result.
The performances of the CrCNN library described in Chapter 6 will be reported.

7.1 Experimental Setup

All the experiments reported in this Chapter are run on... (caratteristiche server).
Used languages: C++, Python...
Used libraries: SEAL...

7.2 MNIST Dataset

The MNIST [43] is a dataset of handwritten digits images and it is composed by a training set of 60000 examples and a test set of 10000 examples....

7.3 CNN Structure

The starting pre-trained CNN $\dot{\Phi}$ before the approximation step (see Figure 4.1) has 9 layers. Its structure is the following:

1. *Convolution Layer:* The input image is $28 \times 28 \times 1$. The convolution has 20 kernels of size 5×5 with a stride of 2×2. The output of this layer is therefore $12 \times 12 \times 20$.

2. *Average Pooling Layer:* This layer has windows of dimension $2 \times 2 \times 1$ with a stride of 1×1. The output of this layer is therefore $11 \times 11 \times 20$.

3. *Batch Normalization Layer:* This layer normalizes separately each dimension in input and make it have mean 0 and variance 1.

4. *Convolution Layer:* The convolution has 50 kernels of size 3×3 with a stride of 2×2. The output of this layer is therefore $5 \times 5 \times 50$.

5. *ReLu Activation Layer:* This layer applies the ReLu function to each input node.

6. *Average Pooling Layer:* This layer has windows of dimension $2 \times 2 \times 1$ with a stride of 1×1. The output of this layer is therefore $4 \times 4 \times 50$.

7. *Batch Normalization Layer:* This layer normalizes separately each dimension in input and make it have mean 0 and variance 1.

8. *Fully Connected Layer:* This layer fully connects the incoming $4 \cdot 4 \cdot 50 = 800$ nodes to the outgoing 500 nodes.

9. *Fully Connected Layer:* This layer fully connects the incoming 500 nodes to the outgoing 10 nodes.

7.3.1 STEP 1

During the *approximation* (STEP 1 see Section 4.2) phase the CNN $\dot{\Phi}$ becomes Φ. The structure of Φ is the same of $\dot{\Phi}$ except for Layer 5, where the ReLu Activation is substituted with the *Square Activation Layer* that squares the value at each input node. The CNN Φ is trained again.

7.3.2 STEP 2

In the *encoding* (STEP 2 see Section 4.2) phase the Binary Search algorithm described in Section 4.4 is run in order to find suitable encryption parameters n, t and q (see Section 4.1). The details of this run are given in Section 7.5. The algorithm finds $n = 4096$, $q = \{q_1 = 36028797005856769,\ q_2 = 18014398492704769\}$, $t = 2^{29}$, where $\sum_{i=1}^{2} log_2 q_i = log_2 q_1 + log_2 q_2 = 55 + 54 = 109$, that is compliant with the security standards described in Table 4.2 for 128-bit security.
Then each weight in Φ is encoded as a fixed precision floating point number as described in the Fractional Encoding part of Subsection 3.2.7 using $n_i = 64$ bits for the integr part, $n_f = 32$ bits for the fractional part and a base $S = 3$. The produced encoded network $\tilde{\Phi}_{n=4096,q=\{q_1,q_2\},t=2^{29}}$ has the same structure of the one in input but it is able to predict over encrypted images, provided

that they are encrypted using the same n, t and q (see Subsection 3.2.4). The encoding of the CNN and the encryption of the data is achieved through the appropriate methods of the CrCnn library described in Section 6.3.

7.3.3 STEP 3

The *testing* (STEP 3 see Section 4.2) is carried over all the 10000 test images of the MNIST Dataset. The parameters found by the Binary Search are valid under the assumption that a re-encryption step is introduced (as explained in Section 7.5) . This intermediate step is put before layer 7.

7.4 Timings And Transformation's Effects

7.4.1 Timings

The *testing* of the encoded CNN $\tilde{\Phi}_{n,q,t}$ using the CrCNN library gives the following results in terms of timings.

Encryption. Each input image is $28 \times 28 \times 1$ (see Section 7.2) and it is encrypted as a ciphertext3D (see method encryptImage in Section 6.3) where each pixel is a ciphertext. It takes on average 2.74 seconds to encrypt an image.

Decryption. The decryption procedure (see method decryptImage in Section 6.3) gets as input a $1 \times 10 \times 1$ ciphertext3D and outputs a floatCube of the same dimensions. It takes on average 0.02 seconds to compute this decryption.

Table 7.1: Testing times of $\tilde{\Phi}_{n,q,t}$

Layer #	**Operation**	Threads' #	**Time (s)**
Layer 1	Convolution $(5 \times 5 \times 20)$	20	30.73
Layer 2	Average Pooling $(2 \times 2 \times 1)$	1	2.45
Layer 3	Batch Normalization	1	2.03
Layer 4	Convolution $(3 \times 3 \times 50)$	50	7.89
Layer 5	Square	50	0.65
Layer 6	Average Pooling $(2 \times 2 \times 1)$	1	0.76
Re-encryption	Decryption+Encryption	1	3.20
Layer 7	Batch Normalization	1	0.68
Layer 8	Fully Connected $(800, 500)$	40	18.23
Layer 9	Fully Connected $(500, 10)$	50	2.45

The timings reported in Table 7.1 are obtained by running the forward on an encrypted image. Since the CrCNN library (see Section 6.3) allows to split the computational load, for some layers this possibility is used to obtain better

performances. The Convolution Layer 1 is run with 20 threads in parallel, this means that each thread performs a single one dimensional convolution with a kernel of dimensions 5×5. The Convolution Layer 4 is run with 50 threads in parallel and this means that each thread performs one convolution of dimensions 3×3 over an input image of $11 \times 11 \times 20$. The Square Layer 5 is run with 50 threads and this means that each one of them computes the operation over one channel in input. For the Fully Connected Layer 8 each thread performs the operation over 20 input neurons (i.e. 800/40), while in the last Fully Connected Layer each thread performs the operation over 10 (i.e. 500/50) input neurons. The most computationally intensive layers remain the two Convolution Layers and the first Fully Connected Layer, while the re-encryption step does not affect too much the forward execution time.

7.4.2 Accuracy Tracking Through Transformation's Steps

During the transformation's steps the CNN is subject to redefinitions that could cause a loss in accuracy as already exposed in Section 4.2. Indeed in the approximation step the ReLu is substituted with a Square function and this introduces some errors in the CNN, since negative neurons are not truncated to be zero as the ReLu does but become small numbers around zero. The Table 7.2 shows that this approximation step causes the starting accuracy of 98% to decrease of 1 point percentage, that is not a big performance's loss. The last transformation step that involves the encoding and thus the application of the CNN to encrypted images does not affect performances at all. This implies that the parameters found by the Binary Search algorithm are optimal for the tested images.

Table 7.2: CNN's accuracy mutation

Accuracy		
Starting $\dot{\Phi}$	**Approximated Φ**	**Encoded $\tilde{\Phi}_{n=4096,q=\{q_1,q_2\},t=2^{29}}$**
98%	97%	97%(**NB da verificare**)

7.5 Binary Search Results

The Binary Search Heuristic described in Section 4.4 is run to find the optimal parameters n, t and q.
Additionally to what explained in Section 4.4, as input to the Binary Search it is possible to set a *maximum number of re-encryption steps* that one can accept to perform during the forward phase. This increases the probability to find feasible encryption parameters and thus to fall in the best case (see Figure 4.3(a)). The binary search eventually outputs also the best point in the CNN where to perform this/these re-encryption step/s. The results showed by the plot in Figure 7.1 are obtained using the following input parameters:

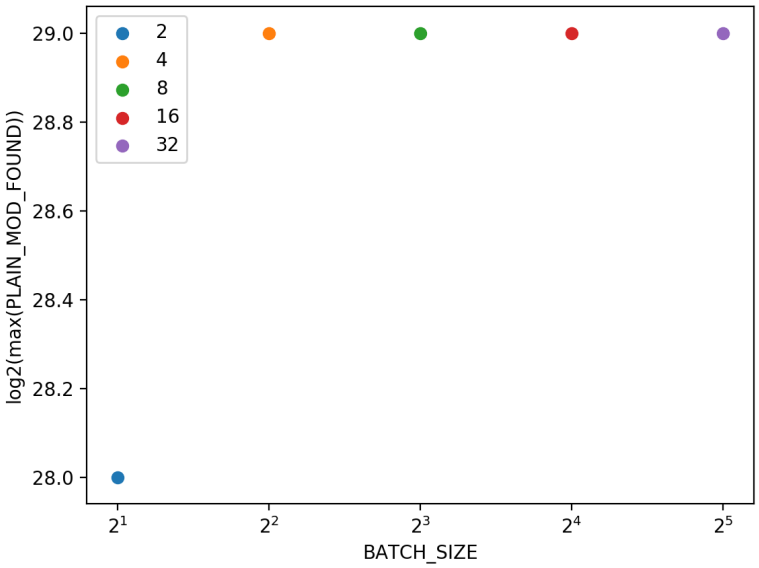


Figure 7.1: Plain Modulus found by the Binary Search for increasing number of images tested ($|D|$)

- Pre-trained approximated model Φ (see Subsection 7.3.1).
- Sorted list of possible plaintext moduli $t_{list} = [2^1, 2^2, ..., 2^{59}]$, with $t_{min} = 23$ and $t_{max} = 33$, that is, the plaintext modulus t can be a power of two between 2^{24} and 2^{34}.
- Security level λ of 128 bits.
- Coefficient modulus q_0 is set as the default value given by the SEAL library for a degree of the polynomial modulus $n = 4096$ and the security level $\lambda = 128$ bits.
- Maximum number of re-encryption steps is set to 1.

The algorithm is run for 50 times and at each run a number of randomly sampled images $|D|$ of 2, 4, 8, 16, 32 (batch size) is tested. In the end there are 50 samples (i.e. obtained plaintext moduli) for each batch size. The plot shows the maximum value of plaintext modulus obtained for a given batch size. One can notice that just using a batch size of 4 images, the binary search is able to find the optimal plaintext modulus $t = 2^{29}$. The binary search sets the point in the CNN for the re-encryption as the point before Layer 7, as showed by Table 7.1.

Chapter 8

Conclusions and Future Works

8.1 Conclusions

8.2 Future Works

Bibliography

[1] GDPR text:. `http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L:2016:119:FULL`.

[2] Cesare Alippi, Simone Disabato, and Manuel Roveri. Moving convolutional neural networks to embedded systems: The alexnet and vgg-16 case. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '18, pages 212–223, Piscataway, NJ, USA, 2018. IEEE Press.

[3] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. Scalable and secure logistic regression via homomorphic encryption. In Elisa Bertino, Ravi Sandhu, and Alexander Pretschner, editors, *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, New Orleans, LA, USA, March 9-11, 2016*, pages 142–144. ACM, 2016.

[4] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian GjÃžsteen, Angela JÃ€schke, Christian A. Reuter, and Martin Strand. A guide to fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2015:1192, 2015.

[5] Norwegian Data Protection Authority. Artificial intelligence and privacy. `https://www.datatilsynet.no/globalassets/global/english/ai-and-privacy.pdf`.

[6] Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. In *MM&Sec*, 2006.

[7] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption how to encrypt with rsa. 1995.

[8] Florian Bourse, RafaÃ«l Del Pino, Michele Minelli, and Hoeteck Wee. Fhe circuit privacy almost for free. *IACR Cryptology ePrint Archive*, 2016:381, 2016.

[9] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd*

Innovations in Theoretical Computer Science Conference, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[10] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[11] S. K. Mitra C. R. Rao. Generalized inverse of matrices and its applications. In *New York:Wiley*, 1971.

[12] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, NIPS'08, pages 289–296, USA, 2008. Curran Associates Inc.

[13] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1999.

[14] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *SIAM Journal on Computing*, pages 542–552, 2000.

[15] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. Technical report, February 2016.

[16] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Manual for using homomorphic encryption for bioinformatics. *Proceedings of the IEEE*, 105(3):552–567, 2017.

[17] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[18] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, August 2014.

[19] David Eisenbud. Commutative algebra: with a view toward algebraic geometry. volume 150. Springer Science & Business Media, 1995.

[20] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

[21] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[22] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[23] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the aes circuit. In *Advances in Cryptology - Crypto 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.

[24] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, pages 75–92. Springer, 2013.

[25] Oded Goldreich. Secure multi-party computation, 1998.

[26] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *Proceedings of the 15th International Conference on Information Security and Cryptology*, ICISC'12, pages 1–21, Berlin, Heidelberg, 2013. Springer-Verlag.

[27] Shai Halevi and Victor Shoup. Design and implementation of a homomorphic-encryption library. manuscript. `http://people.csail.mit.edu/shaih/pubs/he-library`.

[28] David Harvey. Faster arithmetic for number-theoretic transforms. *CoRR*, abs/1205.2926, 2012.

[29] Shohei Kuri, Takuya Hayashi, Toshiaki Omori, Seiichi Ozawa, Yoshinori Aono, Le Trieu Phong, Lihua Wang, and Shiho Moriai. Privacy preserving extreme learning machine using additively homomorphic encryption. In *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017, Honolulu, HI, USA, November 27 - Dec. 1, 2017*, pages 1–8. IEEE, 2017.

[30] Kim Laine. Simple encrypted arithmetic library 2.3.1. `https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf`.

[31] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1219–1234, New York, NY, USA, 2012. ACM.

[32] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

[33] Vincent Migliore, Guillaume Bonnoron, and Caroline Fontaine. Determination and exploration of practical parameters for the latest Somewhat Homomorphic Encryption (SHE) Schemes. working paper or preprint, October 2016.

[34] C. Orlandi, A. Piva, and M. Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP J. Inf. Secur.*, 2007:18:1–18:10, January 2007.

[35] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[36] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IACR Cryptology ePrint Archive*, 2017:715, 2017.

[37] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.

[38] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[39] Saeed Samet and Ali Miri. Privacy preserving id3 using gini index over horizontally partitioned data. In *Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications*, AICCSA '08, pages 645–651, Washington, DC, USA, 2008. IEEE Computer Society.

[40] Saeed Samet and Ali Miri. Privacy-preserving back-propagation and extreme learning machine algorithms. *Data Knowl. Eng.*, 79-80:40–61, 2012.

[41] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1310–1321. ACM, 2015.

[42] Yizhi Wang, Jun Lin, and Zhongfeng Wang. An efficient convolution core architecture for privacy-preserving deep learning. In *IEEE International Symposium on Circuits and Systems, ISCAS 2018, 27-30 May 2018, Florence, Italy*, pages 1–5. IEEE, 2018.

[43] Corinna Cortes Yann LeCun and Christopher J.C. Burges. The mnist database of handwritten digits. `http://people.csail.mit.edu/shaih/pubs/he-library`.

[44] Z. Zhang, J. Wu, D. Yau, P. Cheng, and J. Chen. Secure kalman filter state estimation by partially homomorphic encryption. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, volume 00, pages 345–346, Apr 2018.