Functional encryption applied to privacy-preserving classification : practical use, performances and security

Damien Ligier

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : informatique

Par

Damien Ligier

Functional encryption applied to privacy-preserving classification:

Practical use, performances and security

Thèse présentée et soutenue à Palaiseau, le 15 octobre 2018
Unité de recherche : Lab-STICC
Thèse N° : 2018IMTA0040

Rapporteurs avant soutenance :

Patrick Bas	Directeur de recherche CNRS CRISTAL Lab
Dario Fiore	Assistant research professor IMDEA

Composition du Jury :

Président :	David Pointcheval	Directeur de recherche CNRS, ENS
Examinateurs :	Patrick Bas	Directeur de recherche CNRS CRISTAL Lab
	Dario Fiore	Assistant research professor IMDEA
Dir. de thèse :	Caroline Fontaine	Chargée de recherche CNRS/Lab-STICC et IMT Atlantique
Co-dir. de thèse :	Renaud Sirdey	Directeur de recherche CEA LIST/L3S
Encadrant de thèse :	Sergiu Carpov	Cheucheur CEA LIST/L3S

Functional encryption applied to privacy-preserving classification: practical use, performances and security

Damien Ligier

Contents

Chapter 1

Introduction

Classification algorithms, and more generally machine learning, have proven themselves to be very powerful tools for identifying useful information in large datasets. Since we are now living in the big data era, they help us extracting information from huge amounts of collected data. However it raises concerns about privacy, which brought to the forefront the challenge of designing secure machine learning algorithms able to preserve data confidentiality. Privacy is an important goal of cryptography, so it makes sense to take advantage of it for preserving data privacy in machine learning algorithms.

Context. In the cloud computing era, massive amounts of collected data bring onto the agenda the issue of its privacy. One solution would be to use privacy preserving computation systems, but we desperately lack such efficient systems supporting a large class of functionalities. It would for instance allow an email server to filter spam messages without decrypting anything or, detect criminal faces on encrypted public video surveillance. One of the most important application would be in the medical world. Hospitals and research centers may be legally prevented from sharing their medical databases which in turn precludes them from exploiting different kind of data mining techniques over aggregated datasets. Note that classification algorithms are essential in the aforementioned situations, either medical or not. Hence, we could in fact use privacy preserving computation systems to bring some privacy back in many applications of our daily life.

Data privacy may seem to conflict with other important properties such as the ability to learn and predict or to authenticate, but we really need systems that guarantee more than just privacy. Designing such systems is very challenging. In the literature, there are some constructions regarding authentication, such as the privacy preserving authentication and access con-

trol scheme [RLKD06] based on blind signature and hash chain. This scheme proposed by Ren *et al.*, achieves both security and privacy and has at the same time a highly flexible and light weight authentication and session key establishment protocol. *Lu et al.* designed a different one which has a strong and lightweight RFID privacy authentication protocol [LHH+07] allowing valid readers to explicitly authenticate their dominated tags without leaking private information.

We already mentioned the need for algorithms that would be able to classify encrypted data, but to train a model while keeping the training dataset private is also very important. We do not investigate the last one in this thesis but cite a few constructions from the literature. Chaudhuri and Monteleoni designed a privacy-preserving logistic regression algorithm [CM09] that has a bounded sensitivity of regularized logistic regression and a perturbed classifier. We can also cite Mohan *et al.* who presented a platform [MTS+12] allowing organizations to delegate external aggregate analysis on their datasets, while ensuring that the data analysis is performed in a differentially private manner. Shokri and Shmatikov have designed, implemented and evaluated a practical system [SS15] based on parallelized and asynchronous learning algorithms. It enables multiple parties to jointly learn an accurate neural network model for a given objective, without sharing their input datasets. Abadi *et al.* developed new algorithmic techniques for learning, and a refined analysis of privacy costs within the framework of differential privacy [ACG+16]. There is an important tradeoff between privacy and learnability. It offers many different settings and the community is still focusing on designing solutions with better privacy and better learnability.

The following use case illustrates what one could do with a privacy preserving classifier, which is the focus of this thesis. We assume that a pharmaceutical company has constructed a classifier that takes as input medical and private pieces of information. The company also does not want to divulge its secret classifier. However, it wants to conduct a study on real human data, for example the patients of a hospital. The law about patient medical data disclosure can be very restrictive depending on the country. For instance in France, a hospital cannot share private data of its patients without a slow and cumbersome administrative process. A privacy preserving classifier does the job. A trusted third party generates the public key and the secret keys associated with the company classifier. The hospital encrypts the data of its patients with the pubic key and sends it to the company. The company decrypts them with the secret keys provided by the trusted third party. After decryption, the company only gets the result of the classification which involves the patients data. It is important to notice that the critical data remains encrypted during the whole process. Proceeding in this way, it

can perform its study on real medical data without endangering patient data privacy.

Cryptographers are determined to address cloud computing era concerns. From a theoretical point of view, a perfect solution is to compute over encrypted data without having to decrypt. There was a breakthrough with the emergence of pairing-based and lattice-based constructions, but the more general they are, the less practical. There are several primitives such as among others, fully and/or somewhat homomorphic encryption [AGH10, VDGHV10, BGV12, FV12, BV14, BLLN13, GSW13, AP14, DM15, BR15, CGGI16], functional encryption [GKP+13, GGH+13], attribute based encryption [Wat11, LYZ+13] or searchable encryption [VLSD+10].

In this thesis we focus on *Functional Encryption* (FE) [SW05, BSW11, BCFG17] which is a quite recent generalization of public-key cryptography. Within this paradigm, an authority allows users to partially decrypt ciphertexts. Therefore, it enables fine-grained access control to encrypted data but also limits the computations which can be performed on the ciphertexts. The authority of the system generates secret keys associated with functions. For example (cf. figure 1.1) let sk_f be a secret key associated with function f and ct_x be an encryption of x, the output of the decryption algorithm taking sk_f and ct_x as inputs is $f(x)$ (rather than x as in an usual public-key encryption system).

The holy grail of this domain is to design a scheme that enables to derive a secret key sk_f for any polynomial time computable function f. Regretfully, up today literature provides practical functional encryption schemes for linear (i.e. inner-product) and quadratic polynomial evaluation functionalities only.

Contribution. In this thesis we focus on a specific architecture of privacy preserving classification, which is based on the use of functional encryption. More precisely, we propose a combination of functional encryption and machine learning in order to perform classification over encrypted data. Our construction is generic to any functional encryption scheme but we needed to observe it in a real life scenario. This is why we chose to mainly focus on the inner-product functional encryption, the only functional encryption scheme known to be practical when this thesis began. We also experimented and then compared it with quadratic polynomial functional encryption, which has been later proposed in the literature. We then placed ourselves in an attacker shoes and tried to exploit the construction we had proposed. Our goal at this moment was to break its privacy preserving property. We propose different ways to attack use cases using machine learning and functional encryption. Finally, we study how to avoid this kind of attacks and propose

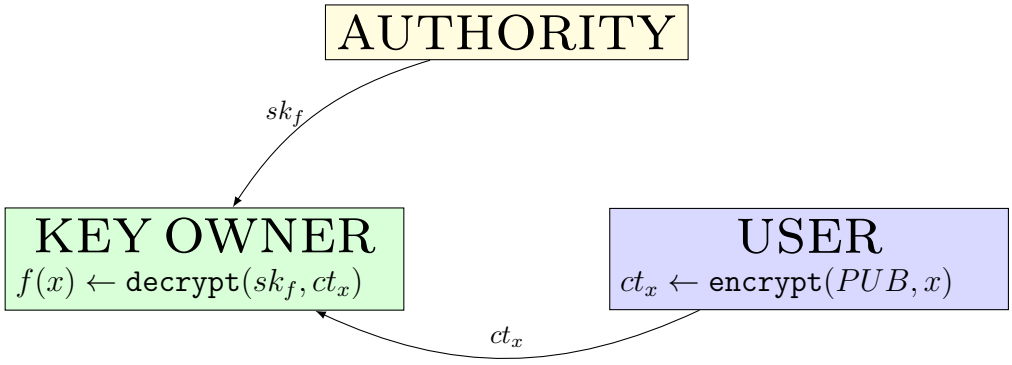


Figure 1.1: An overview of a functional encryption system.

a way to manage the trade-off between the performance of the privacy preserving classification and the efficiency of the attack.

Overview. The first part of this thesis is dedicated to the state of the art. We first recall the context of public-key cryptography and then move to functional encryption. We describe what is different in a private-key setting, but also its security, what is making it different to fully homomorphic encryption and finally its notion of leakage. At this point, we focus on a particular instance of functional encryption: inner-product functional encryption. We describe two constructions from the literature based on the decisional Diffie-Hellman assumption or on the learning with error assumption. In this thesis, machine learning techniques are used, so, we introduce the required background on classification algorithms. We begin with linear classification, moving on to extremely randomized trees classifiers, and finally artificial neural networks. We also describe the principal component analysis and the MNIST dataset of handwritten digits that we used all along this thesis.

The second part of this thesis is for our contributions. We begin by presenting our construction of a privacy-preserved classification system. We begin by giving details about our implementation of the inner-product functional encryption scheme from [ALS16]. Then we expose different ways to use inner-product functional encryption to achieve privacy preserving classification: using a linear classifier, an extremely randomized trees classifier or a neural network classifier. We also detail the setup of our experimentations using the MNIST dataset. We then do the same experimentation but with a quadratic polynomial functional encryption scheme instead of the inner-product functional encryption one and give some results.

The next chapter focuses on how to attack the previously introduced

privacy preserving classification scheme. We give the context of the attack and then describe three attack instantiations: one based on the principal component analysis, other based on fully connected neural networks, and the last one based on convolutional neural networks. An analysis of the practical experimentation results of those attacks on the MNIST dataset follows. Finally, we discuss its generalization to other functional encryption schemes.

The last chapter of the second part aims to study the trade-off between the performances of the classification and the efficiency of the attacks, in a context of privacy-preserving classification based on inner-product functional encryption. We recall how we get those vectors associated with the functional encryption secret keys, and then propose two ways to get different ones: generating randoms vectors, and using a generative adversarial network in different settings. We finally provide an extensive discussion on the results we obtained.

Part I

State of the art

Chapter 2

Functional Encryption

Functional encryption (FE) is a generalization of the traditional public key cryptography that enables a fine-grained access control for encrypted data. We introduce in this chapter the cryptographic notions used in this thesis. We start by traditional public key cryptography, and two computational hardness assumptions which the security of cryptographic primitives we use will rely on. Then, we describe what are functional encryption schemes. We then focus on inner-product functional encryption, its private-key setting, its security, but also what makes it very different from fully homomorphic encryption. In particular, we introduce two particular designs of functional encryption for the inner-product functionality. Finally, we look at functional encryption beyond the inner-product functionality.

2.1 Public key cryptography

During the 70's, Diffie and Hellman [DH76b, DH76a] introduced a radically new concept of public-key cryptography. It depicts a secure communication between two parties without having them to share a secret (symmetric-key cryptography), which was the only trusted solution back then. Since that, we have at our disposal several public-key encryption schemes. They are based on a few computational problems as the hardness of factoring or the hardness of computing discrete logarithms [DH76a, RSA78, McE78, AD97].

Today, public-key cryptography is widely present in our lives to the point where it is impossible to envision current electronic commerce security without it. Indeed, symmetric-key cryptography and public-key cryptography complete each other: the heavier public-key cryptography is used first to achieve authentication but also to provide both parties the same temporary secret used for the second part of symmetric-key communication.

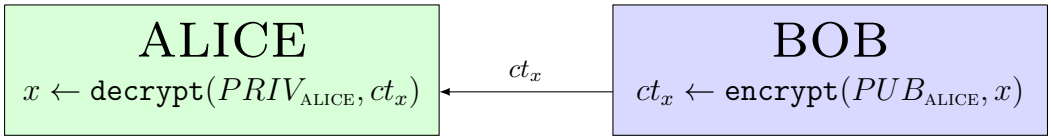


Figure 2.1: An overview of public-key cryptography. In this figure the public and the private key are respectively called PUB and PRIV.

Figure 2.1 illustrates how public-key encryption works. Alice had already generated her keys (the private key $PRIV_{\text{ALICE}}$ and the public-key PUB_{ALICE}) and had published the public one. Bob uses her published key to encrypt a message x into a ciphertext ct_x. When Alice gets ct_x from Bob, she can decrypt it with her private key $PRIV_{\text{ALICE}}$ and gets the message x. Note that she is the only one able to decrypt it.

A computational hardness assumption states that a particular problem cannot be solved efficiently, i.e. "in polynomial time". Cryptographers use those assumptions to proove cryptographic scheme security. In this thesis we will refer to some of them such as the Decisional Diffie-Hellman assumption, the computational Diffie-Hellman assumption or the Extended learning with errors assumption. We introduce them in the next section.

2.1.1 Decisional Diffie-Hellman assumption

The *Decisional Diffie-Hellman* assumption (DDH) has been used to prove that the Diffie-Hellman protocol [DH76a] was useful for practical cryptographic purposes. It speaks about groups and in practice those are finite groups. The *computational Diffie-Hellman* assumption (CDH) which states that in a particular group, no efficient algorithm can compute g^{ab} with only g, g^a, g^b, was not sufficient. The DDH assumption is way stronger than the CDH assumption. Assuming that DDH stands in a group means that there is no efficient probabilistic algorithm that given any triplet g^a, g^b, g^c outputs "true" if $a = bc$ and "false" otherwise. We note that there exist groups where DDH assumption is false but for which the CDH assumption is believed to be true. We now give the formal definition of the DDH assumption.

For more details about CDH and DDH please refer to [Bon98, Gal12].

Definition 1. *[Bon98] In a cyclic group $\mathbb{G}$ of prime order q, the* Decisional Diffie-Hellman (DDH) *problem is to distinguish the distributions*

$$D_0 = \{(g, g^a, g^b, g^{ab}) | g \xleftarrow{R} \mathbb{G},\ a, b \xleftarrow{R} \mathbb{Z}_q\}$$
and

$$D_1 = \{(g, g^a, g^b, g^c) | g \xleftarrow{R} \mathbb{G},\ a, b, c \xleftarrow{R} \mathbb{Z}_q\}.$$

2.1.2 Extended learning with errors assumption

The extended learning with error problem has been introduced by O'Neil, Peikert and Waters [OPW11]. It can be described as $\text{LWE}_{\alpha,q}$, with a number m fixed. The problem is to distinguish between the following two distributions (where z is sampled from a specific distribution):

$$D_0 = \{(A, A \cdot s + e, z, \langle e, z \rangle) | A \hookleftarrow \mathbb{Z}_q^{m \times n}, s \hookleftarrow \mathbb{Z}_q^n, e \hookleftarrow D_{\mathbb{Z},\alpha q}^m\}$$
$$\text{and}$$
$$D_1 = \{(A, u, z, \langle e, z \rangle) | A \hookleftarrow \mathbb{Z}_q^{m \times n}, u \hookleftarrow \mathbb{Z}_q^m, e \hookleftarrow D_{\mathbb{Z},\alpha q}^m\}.$$

Many variants of the learning with error problem exist. One of those is the multi-hint extended-LWE problem [ALS16]. Its definition follows.

Definition 2. *[ALS16] The multi-hint extended-LWE problem is a variant of extended-LWE problem for which multiple hints are given for the same noise term. It exists a reduction from LWE to mheLWE [ABDCP15]. It is formalized as follows. Let q, m, t be integers, α be a real and τ be a distribution over $Zt \times m$, all of them functions of a parameter n. The multi-hint extended-LWE problem $\text{mheLWE}_{q,\alpha,m,t,\tau}$ is to distinguish between the distributions of the tuples*

$$(A, A \cdot s + e, Z, Z \cdot e) \text{ and } (A, u, Z, Z \cdot e) ,$$

where $A \hookleftarrow \mathbb{Z}_q^{m \times n}$, $s \hookleftarrow \mathbb{Z}_q^n$, $u \hookleftarrow Z_q^m$, $e \hookleftarrow D_{Z,\alpha q}^m$, and $Z \hookleftarrow \tau$.

2.2 Introduction to Functional encryption

Functional encryption (FE) is a generalization of traditional public key cryptography. Obviously, there is a *public key* required to encrypt messages. But it offers the possibility to decrypt ciphertexts partially with a fine-grained control. Its design makes it well suited for purposes such as cloud computing [BSW11] or verifiable computation [PRV12] among others.

FE requires a new party, called *the authority*, but also an algorithm called *key generation*. The authority generates and keeps the *master secret key* MSK along with the public key. This particular key is necessary to derivate what are called *secret keys* (using the key generation algorithm), which are given to users and must stay secret. Those secret keys are associated with functions, for instance, we denote by sk_f the secret key associated with

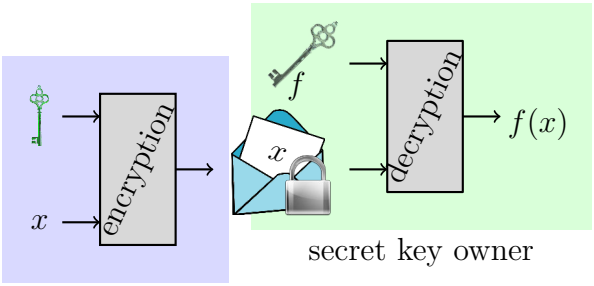


Figure 2.2: An overview of FE systems. The green key represents the public key of the system, the silver key is a secret key associated with the function f. The envelope with the padlock represents a ciphertext.

function f. Let ct_x be an encryption of a message x. When a user owning sk_f and ct_x computes the decryption algorithm with sk_f and ct_x as inputs, he gets $f(x)$ as output. This is not a traditional way to decrypt, but it can be considered as an evaluation of f over encrypted messages, giving an unencrypted output. Figure 2.3 illustrates the system described above.

By "partially decripted" we mean that the output of the decryption algorithm is not the plaintext x but $f(x)$, which will reveal partially x.

Boneh *et al.* give in [BSW11] the following standard definitions for functional encryption using the notion of functionality. Note that the previous notation is the one used in this thesis and they do not match with the description that follows (f is renamed $F(K, \cdot)$).

Definition 3. *A* functionality F *defined with* (K, X) *is a function* $F : K \times X \to \Sigma \cup \{\perp\}$. *The set* K *is the key space, the set* X *is the plaintext space, and the set* Σ *is the output space and does not contain the special symbol* $\perp$.

Definition 4. *A* functional encryption *scheme for a functionality* F *is a tuple* $\mathcal{FE} = (\texttt{setup}, \texttt{keyGen}, \texttt{encrypt}, \texttt{decrypt})$ *of four algorithms with the following properties.*

- *The* `setup` *algorithm takes as input the security parameter* 1^λ *and outputs a pair of a public key and a master secret key* (PUB, MSK).
- *The* `keyGen` *algorithm takes as inputs the master secret key* MSK *and* $k \in K$ *which is a key of the functionality* F. *It outputs a secret key* sk *for* k.
- *The* `encrypt` *algorithm takes as inputs the public key* PUB *and a plaintext* $x \in X$. *This randomized algorithm outputs a ciphertext* c_x *for* x.

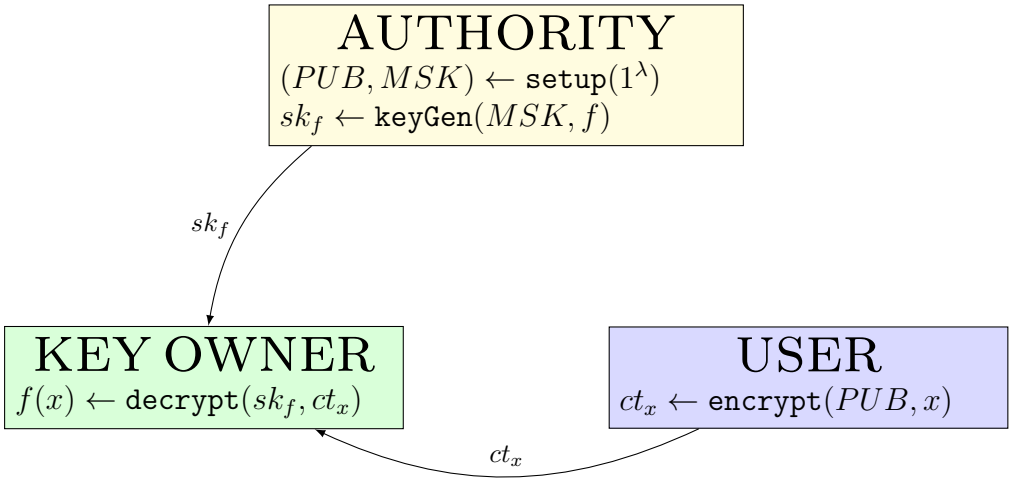


Figure 2.3: The three actors of a functional encryption system, the algorithm they use and their communications. In this figure, the public key, the master secret key and the secret key associated with the function f are respectively called PUB, MSK and sk_f.

- *The* `decrypt` *algorithm takes as inputs the public key* PUB*, a secret key and a ciphertext. It outputs* $y \in \Sigma \cup \{\perp\}$*.*

It is required that for all $(PUB, MSK) \leftarrow \texttt{setup}(1^\lambda)$*, all keys* $k \in K$ *and all plaintexts* $x \in X$*, if* $sk \leftarrow \texttt{keyGen}(MSK, k)$ *and* $c \leftarrow \texttt{encrypt}(PUB, x)$ *we have* $F(K, X) = \texttt{decrypt}(PUB, sk, c)$ *with an overwhelming probability.*

We do not follow those notations in this thesis. For instance the decryption algorithm explicitly requires the public key in the previous definition, however, we will use a decryption algorithm notation that implicitly takes as input the public parameters (public key) of the system (as in figure 2.3).

The cryptographic community is looking for public-key functional encryption schemes enabling to evaluate any polynomial time computable function. Goldwasser *et al.* proposed a construction based on fully homomorphic encryption [GKP+13], Garg *et al.* proposed another construction using an indistinguishability obfuscator [GGH+13]. At present, however, these constructions remain mostly of theoretical interest. Nevertheless, more recently schemes for simpler functionalities have been proposed. For example *inner-product functional encryption* (IPFE) [ABDCP15, ALS16] or *functional encryption for quadratic functions* [BCFG17].

Functional encryption generalizes among others, attribute-based encryption [Wat11, LYZ+13, Wat05], identity based encryption [BF01, SW05] or predicate encryption [KSW08, OT09]. Figure 2.4 shows it with more details.

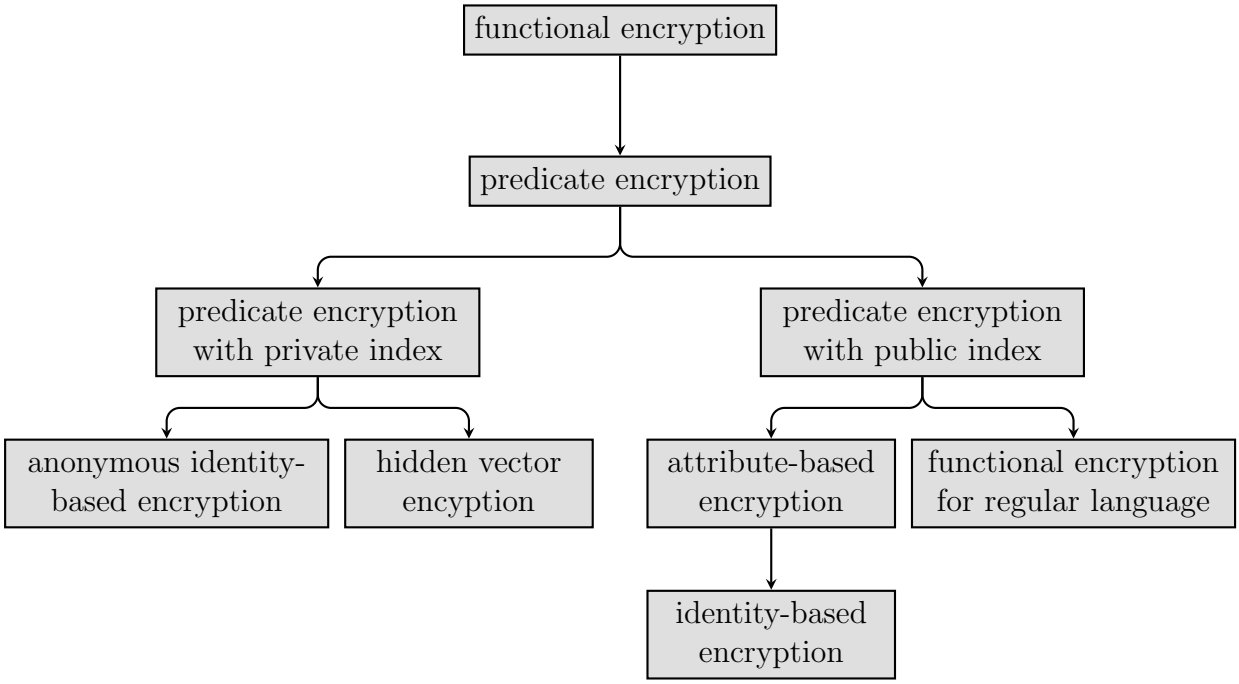


Figure 2.4: An overview of what is generalized by functional encryption.

Those primitives are also public key encryption schemes that perform some light computation over ciphertexts.

2.2.1 Public-key setting versus private-key setting

Because of the public-key setting, every constant in the function associated with a secret key is known by its owner. Indeed, since Bob is able to encrypt every message x_i with Alice's public key, and also to decrypt all of them with his secret key sk_f associated with the function f, he gets an unlimited number of couples $(x_i, f(x_i))$ which enables him to determine the constants of f. On the contrary, with a *private-key setting*, it is possible to have hidden values in secret key functions. There is, of course, no public key in such systems, and the encryption algorithm takes instead as input the master secret key. When it is about private-key functional encryption, we can find in the literature some inner-product functional encryption schemes [BS15, SSW09, BRS13], but also some multi-input inner-product functional encryption schemes [BKS16, DOT18, KS17]. This thesis focuses on public-key functional encryption.

2.2.2 Functional encryption's security

A fundamental security requirement for functional encryption schemes is called *collusion resistance*. Indeed, the authority of a functional encryption scheme delivers several secret keys associated with different functions (authorized function evaluation) to the users. The idea behind collusion resistance is that if a user owns different secret keys $\{sk_{f_i}\}_i$ and an encryption of x, he cannot learn about x more than $\{f_i(x)\}_i$ (the set of the evaluation outputs). This property is captured by two security definitions: *indistinguishability-based security* and *simulation-based security*.

The idea behind *indistinguishability-based security* (IND) is that an attacker is given two messages x_0 and x_1 such that, for all the secret keys $\{sk_{f_i}\}$ he owns, $f_i(x_0) = f_i(x_1)$. Then, when he gets an encryption of x_b, he has to determine if $b = 0$ or if $b = 1$. It shows how a single message can remain secure against an arbitrary number of users performing a collusion. However, it appears that the IND definition is not strong enough: indeed, it has been shown [BSW11, O'N10] that a trivially insecure scheme can be proved IND-secure.

The *simulation-based security* (SIM) means that a secret key associated with a function f makes it only possible to get $f(x)$ from an encryption of x. Some results [BSW11, BO13, AGVW13] show that the SIM definition is not always achievable. To sum up, the SIM-based security is has real security guarantee but might be impossible to achieve for som cases, and the IND-based security does not prevent a scheme from being insecure.

2.2.3 Difference between functional encryption and fully homomorphic encryption

Traditional public-key cryptography has been generalized in many ways, among others, *Functional Encryption* (FE) but also *Fully Homomorphic Encryption* (FHE) [G+09, AGH10, VDGHV10, BGV12, Bra12, FV12, GSW13, BGV14, CGGI16, BDF18]. Both enable to compute algorithms over encrypted inputs but one difference between them is that FE's decryption algorithm output is unencrypted, when FHE one's remains encrypted. Figure 2.5 shows an overview of an FHE system.

There is no need of a trusted authority within FHE systems. It works as a traditional public key system except that there are two additional algorithms: *add* and *mul*. The first one takes two ciphertexts c_m and $c_{m'}$ of two plaintexts m and m', and it returns a new ciphertext, which is the encryption of the message $m + m'$. The second algorithm takes also c_m and $c_{m'}$ bur returns a new ciphertext, which is the encryption of the message $m \times m'$. It means

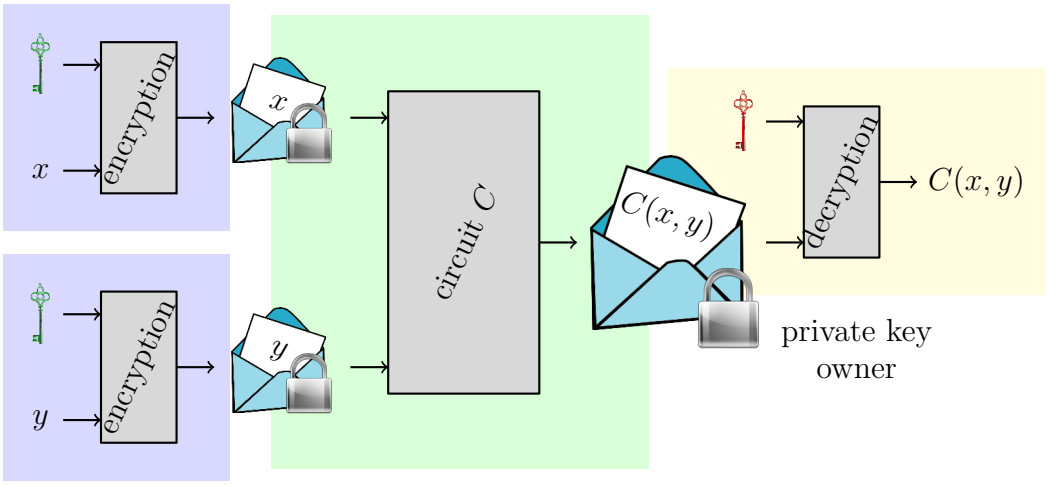


Figure 2.5: An overview of FHE systems. The green key represents the public key of the system, the red one is the private key of the system. The circuit C is composed of homomorphic gates addition and multiplication. The envelopes with a padlock represents ciphertexts.

that it is possible to perform algorithms over encrypted data.

Another difference between FE and FHE is that the first one enable the evaluation of any circuit over encrypted data when the second one enable only the computation of some functions. It means that one can compute what he wants using *add* and *mul* algorithms on encrypted data but will never have access to the result. In a FE context, algorithms that can be computed are fixed thanks to the secret keys and cannot be modified. Note that the output is unencrypted.

Concerning keys, FHE follows the traditional public key cryptography with a public key and a secret key, but the FHE renamed the secret key in master secret key and uses it to generate secret keys (associated with some functions) that end up being given to users.

FHE and FE share some resemblance such as the fact that they perform computation over encrypted inputs. To illustrate this, we can mention that FHE has been used to construct a theoretical FE for all circuits [GGH+13].

2.2.4 Functional encryption's leakage

The main purpose of functional encryption is to provide something more "in between" than the traditional "all or nothing" decryption in cryptography.

It enables a fine-grained partial decryption of cyphertexts. When one knows about the context of a FE system, it may be possible to use it to get closer to the "all" decryption than he should be. This leakage is independent of the cryptographic system security. Note that in a FE system, functions (associated with the secret keys) may be computed over every plaintext, so as a consequence all plaintexts share similarities. Those shared similarities could be used to build approximations of messages that have been encrypted. This is one of our concerns in this work.

2.3 Inner-product functional encryption

Functional encryption schemes that enable the evaluation of inner products [ABDCP15, ALS16] are called *functional encryption for the inner-product functionality*, *inner-product functional encryption*, or *inner-product encryption*. In those schemes, secret keys are associated with inner-product functions. Let $\vec{v}, \vec{w}$ be vectors, $sk_{\vec{v}}$ be the secret key associated with the inner-product function $\langle \vec{v}, \cdot \rangle$, and $ct_{\vec{w}}$ be an encryption of $\vec{w}$. The decryption algorithm with $sk_{\vec{v}}$ and $ct_{\vec{w}}$ as inputs, outputs $\langle \vec{v}, \vec{w} \rangle$.

Recently, Abdalla *et al.* [ABDCP15] proposed constructions for the inner product encryption schemes satisfying standard security definitions, under well-understood assumptions: the *Decisional Diffie-Hellman* (DDH) and *Learning With Errors* (LWE). However they only proved their schemes to be secure against *selective adversaries*. Agrawal *et al.* [ALS16] upgraded those schemes to provide them a *full security* (security against *adaptive attacks*). In this work, we focus on these inner product schemes, thus on the fully secure functional encryption for the inner product functionality under the DDH assumption [ALS16].

2.3.1 Inner-product functional encryption based on the DDH assumption

We now recall the algorithms used in the design of functional encryption for inner-product scheme from [ALS16], that provides full security under the DDH assumption. There is no need to recall the scheme security proof in this thesis however it can be found in the paper that introduced it.

Let $\vec{w} = (w_1, \cdots, w_m) \in \mathbb{Z}_q^m$ be the vector we want to associate a key.

Let $\vec{v} = (v_1, \cdots, v_m) \in \mathbb{Z}_q^m$ be a plaintext we want to encrypt.

The decryption algorithm uses a discrete logarithm computation in a large size group (which in general is hard to compute). Coefficients of the plaintext vector $\vec{w}$ and the key vector $\vec{v}$ belong to $\{-\beta, ..., 0, ..., \beta\}$ where β is a small

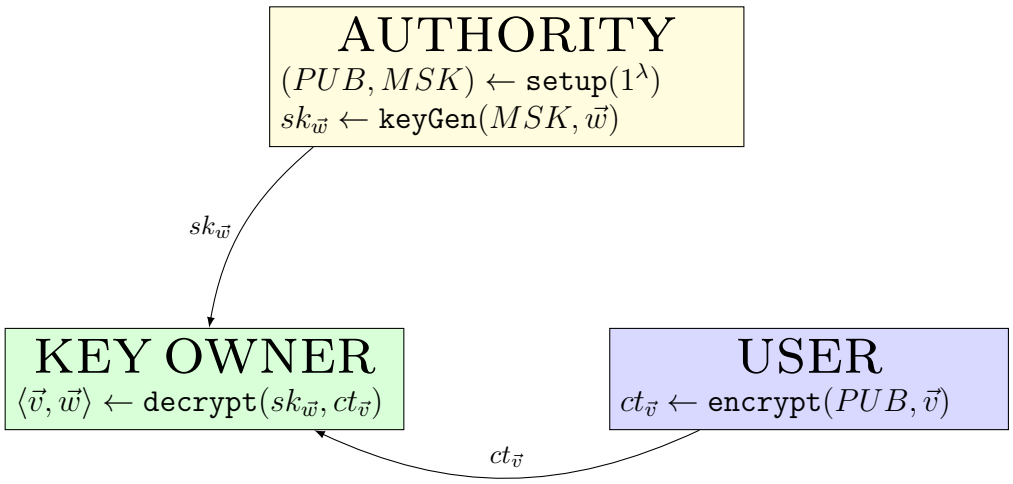


Figure 2.6: The three actors of an inner product functional encryption system, the algorithm they use and their communications. In this figure, the public key, the master secret key and the secret key associated with the vector $\vec{w}$ are respectively called PUB and MSK and $sk_{\vec{w}}$.

Algorithm 1 $\texttt{setup}(1^\lambda, 1^m)$

choose a cyclic group $\mathbb{G}$ of prime order $q > 2^\lambda$ with generators $g, h \in \mathbb{G}$
for all $1 \leq i \leq m$ **do**
    $s_i, t_i \xleftarrow{R} \mathbb{Z}_q$
    $h_i \leftarrow g^{s_i} \cdot h^{t_i}$
$PUB \leftarrow (\mathbb{G}, g, h, \{h_i\}_{1 \leq i \leq m})$
$MSK \leftarrow (\{s_i\}_{1 \leq i \leq m}, \{t_i\}_{1 \leq i \leq m})$
return (PUB, MSK)

Algorithm 2 $\texttt{keyGen}(\text{MSK}, \vec{w})$

$s_{\vec{w}} \leftarrow \sum_{i=1}^{m} s_i \cdot v_i$ ▷ $MSK = (\{s_i\}_{1 \leq i \leq m}, \{t_i\}_{1 \leq i \leq m})$
$t_{\vec{w}} \leftarrow \sum_{i=1}^{m} t_i \cdot v_i$
return $sk \leftarrow (s_{\vec{w}}, t_{\vec{w}})$

Algorithm 3 $\texttt{encrypt}(\text{PUB}, \vec{v})$

$r \xleftarrow{R} \mathbb{Z}_q$ ▷ $PUB = (\mathbb{G}, g, h, \{h_i\}_{1 \leq i \leq m})$
$C \leftarrow g^r$, $D \leftarrow h^r$
for all $1 \leq i \leq m$ **do**
    $E_i = g^{v_i} \cdot h_i^r$
return $ct \leftarrow (C, D, \{E_i\}_{1 \leq i \leq m})$

integer, so the possible interval of $\langle \vec{v}, \vec{w} \rangle$ is small as well. When the output interval of the discrete logarithm is small and known we can use Shank's baby step giant step algorithm [Sha71] to compute it efficiently, simply use a lookup table or a combination of both.

Algorithm 4 `decrypt`(PUB,sk,ct)

1: $E \leftarrow \prod_{i=1}^{m} E_i^{v_i} / (C^{s_v} \cdot D^{t_v})$ ▷ $c = (C, D, \{E_i\}_{1 \leq i \leq m})$
2: $r \leftarrow log_g(E)$ ▷ $sk = (s_{\vec{w}}, t_{\vec{w}})$
3: **return** r

2.3.2 Inner-product functional encryption based on the LWE assumption

Agrawal, Libert and Stehlé [ALS16] proposed a construction of a fully secure inner-product functional encryption based on the learning with errors assumption. More precisely, its security relies on the hardness of a variant of the LWE problem called multi-hint extended-LWE problem, that was first introduced by O'neill, Peikert and Waters [OPW11], and more deeply investigated in [ASP12], [BLP+13]. Details about the LWE problem and its multi-hint variant are given is 2.1.2.

Note that, in this scheme, the inner product is computed modulo p.

We now recall algorithms from this particular inner-product functional encryption based on the LWE assumption [ALS16].

Algorithm 5 `setup`(1^n,1^ℓ,p)

1: set integers m and k, a real $0 \leq \alpha \leq 1$ and a distribution τ over $\mathbb{Z}^{\ell \times m}$
2: set $q = p^k$ and sample $A \hookleftarrow \mathbb{Z}_q^{m \times n}$ and $Z \hookleftarrow \tau$
3: compute $U = Z \cdot A \in \mathbb{Z}_q^{\ell \times n}$
4: set $mpk \leftarrow (A, U)$ and $msk \leftarrow Z$
5: **return** (mpk, msk)

Let $\vec{x} = (x_1, \cdots, x_\ell) \in \mathbb{Z}_p^\ell$ be the vector we want to associate a key, and let st be a state.

Algorithm 6 `keyGen`(msk,$\vec{x}$,st)

1: **if** $\vec{x}$ linearly independent from key queries that have been made so far **then**
2: set $\bar{x} \leftarrow x$
3: set $z_{\vec{x}} \leftarrow \bar{x}^T \cdot \mathbb{Z}^m$ ▷ $msk = Z$
4: add $(\vec{x}, \bar{x}, z_{\vec{x}})$ to st
5: **else**
6: set $\bar{x} \leftarrow \sum_i k_i \bar{x}_i$
7: set $z_x \leftarrow \sum_i k_i z_i$
8: **return** $(\bar{x}, z_x)$

Let $\vec{y} = (y_1, \cdots, y_\ell) \in \mathbb{Z}_p^\ell$ be the vector we want to encrypt. mpk is the public key.

Algorithm 7 `encrypt`(mpk,$\vec{y}$)

1: sample $s \hookleftarrow \mathbb{Z}_p^\ell$, $e_0 \hookleftarrow D^m_{\mathbb{Z},\alpha q}$ and $e_1 \hookleftarrow D^\ell_{\mathbb{Z},\alpha q}$ ▷ $mpk = (A, U)$
2: compute $c_0 \leftarrow A \cdot s + e_0$ and $c_1 \leftarrow U \cdot s + e_1 + p^{k-1} \cdot \vec{y}$
3: **return** $C \leftarrow (c_0, c_1)$

mpk is the public key, $(\bar{x}, z_x)$ is a secret key associated with the vector $\vec{x}$ and C is an encryption on the vector $\vec{y}$.

Algorithm 8 `decrypt`(mpk,$(\bar{x}, z_x)$,C)

1: compute $\mu' \leftarrow \langle \bar{x}, c_1 \rangle - \langle z_x, c_0 \rangle \mathrm{mod} q$ ▷ $C = (c_0, c_1)$
2: compute μ that minimize $|p^{k-1} \cdot \mu - \mu'|$
3: **return** μ

2.4 Functional encryption beyond the inner-product functionality

We can find in the literature other kind of functional encryption schemes. Multi-input inner-product functional encryption [KLM+16, LL16, AGRW17, ACF+17] or quadratic polynomial functional encryption [BCFG17] are pretty close to IPFE schemes and can also be efficient in a practical point of view. The dream of designing a functional encryption scheme for all circuits [GGHZ16a, GKP+13, GVW12, GGH+13] is yet still out of reach in real life scenario. In addition to being not realistic, proposed schemes support sometimes only a

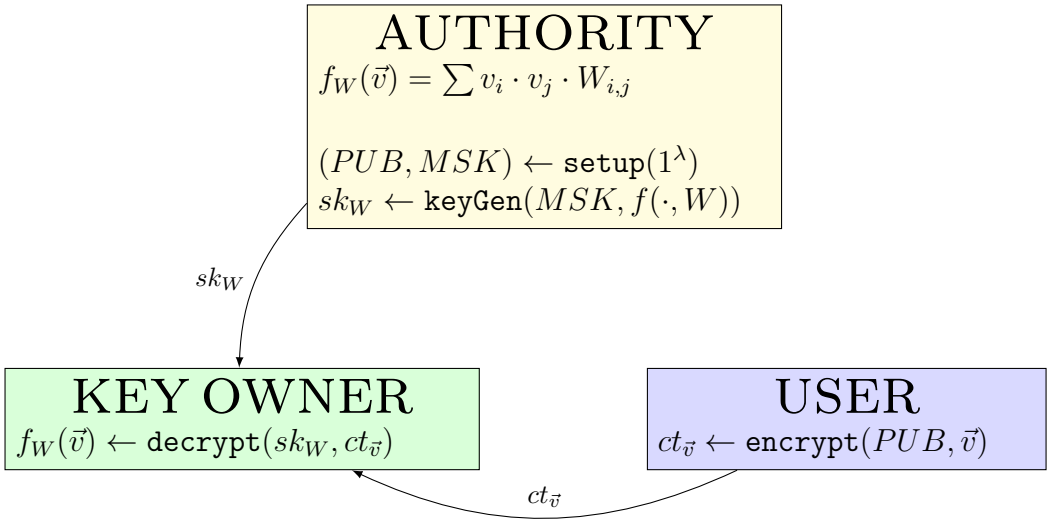


Figure 2.7: The three actors of a functional encryption system for quadratic functions, the algorithm they use and their communications. In this figure, the public key, the master secret key and the secret key associated with the matrix W are respectively called PUB and MSK and sk_W.

bounded number of collusions or have their security based on poorly understood assumptions.

We will describe few of those, as functional encryption for quadratic polynomial, multi-input functional encryption and some theoretical functional encryption for all circuits.

Functional encryption is also a generalization of identity based encryption [BF01, SW05], attribute based encryption [Wat11, LYZ+13, Wat05], predicate encryption [KSW08, OT09] among others. Figure 2.4 shows some of the encryption schemes generalized by functional encryption.

Quadratic polynomial functional encryption

We call functional encryption for quadratic functions a functional encryption system that can evaluate degree-two polynomials [BCFG17]. Let $\vec{v}$ be a vector, $ct_{\vec{v}}$ an encryption of $\vec{v}$, W a square matrix and sk_W the secret key associated with W. The decryption of $ct_{\vec{v}}$ with sk_W returns $\vec{v} \cdot W \cdot \vec{v}^T = \sum v_i \cdot v_j \cdot w_{i,j}$.

Multi-input inner-product functional encryption

Multi-input functional encryption[GGG+14] is a generalization of functional encryption. Let f be a n-ary function. With a multi-input functional encryption one can encrypt n different messages m_1, ... ,m_n into n ciphertexts c_1, ... ,c_n . The authority can generate a secret key sk_f associated with the function f. Then when the decryption algorithm is used with sk_f and c_1,...,c_n, it outputs $f(x_1, ..., x_n)$. Some practical schemes have been proposed for the inner-product functionality[KLM+16, LL16].

Functional encryption for all circuits

Goldwasser et al.[GKP+13] proposed a functional encryption scheme for all circuits using heavy construction blocks as an attribute-based encryption scheme, a fully homomorphic encryption scheme and a garbled circuit.

Garg *et al.* proposed another construction based on non-interactive zero-knowledge protocols and indistinguishability obfuscators [GGH+13]. In [GGHZ16b], Garg *et al.* also proposed a different construction based on graded encoding, branching programs and punctured pseudorandom functions. Nevertheless, these both constructions cannot today lead to practical schemes, as it is still an open problem to build secure versions of some of their parts, as for examples multi-linear maps.

Hence, at present these three temptations to build functional encryption schemes for all circuits remain only of theoretical interest.

Chapter 3

Necessary machine learning background

In this thesis, we are exploiting some of the powerful Machine learning tools. This field gives the ability to a computer to learn a task through data, without being explicitly programmed. It means that the computer will progressively improve its performances on that specific task.

In this chapter we detail a few machine learning basics, as defining supervised and unsupervised machine learning, training and test sets. We also introduce the MNIST dataset of handwritten digits that we used in this thesis. We then focus on classification and detail linear classifiers, principal component analysis and extremely randomized trees. We finally move on to artificial neural networks. We broach convolutional networks and generative adversarial one before going deeper with their activation functions and different measures involved.

3.1 Machine learning basics

Machine learning (ML) [WFHP16] aims for automatic detection of meaningful patterns in data. It provides algorithms enabling program to "learn". The "learn" word refers to the process of converting training data into a program that can perform some tasks. This program represents, in a way, some expertise or knowledge.

The advantages of ML are, among others, the adaptability, the fact that they can be better than a human-written programs, and also easier to conceive than an algorithm by hand.

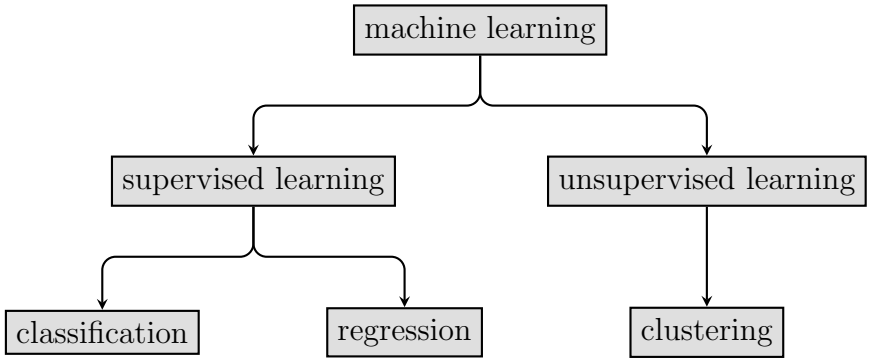


Figure 3.1: A succinct description of ML categories.

3.1.1 Supervised machine learning

Supervised machine learning denotes machine learning algorithms that learn how to map inputs to outputs according to a set of input-output pairs. It is among ML techniques, the one that has the highest economic value.

Those algorithms are split into two successive parts. The first one is the *learning* phase, which analyses the inputs and learns how to make proper predictions on such kind of data. The second part is the prediction phase, where classification of new data is performed using the produced algorithm of the first phase. When the prediction result is a discrete value, we speak about *classification* (class prediction for example), while for a continuous value, we speak about *regression* (temperature prediction for example).

During the training process, the *training set* is used to learn more about the task, i.e. modifies the weights in a way that make the prediction closer to the outputs (or labels). Then, the *validation set* is used to verify that there is not just an improvement for the training set. Finally the *test set* enables to test the actual predictive power of the network.

3.1.2 Unsupervised machine learning

Unsupervised machine learning regroups methods that find hidden structure from data with no labels. There is no possibility to evaluate the accuracy of the algorithms produced. Unsupervised learning includes among others *clustering*. It tries to group objects when they share similarities. Those groups are called clusters.

Figure 3.1 shows how ML and its subcategories can be represented. In this thesis we employ only supervised ML.

3.2 Classification algorithms

Statistical classification identifies which class an object belongs to, using its characteristics. This process is based on the learning phase performed on a training set. Classification use cases are for example the detection of spam emails, speech recognition, handwritten recognition or the elaboration of patient's diagnosis using its medical characteristics.

3.2.1 Linear classification

Linear classification algorithms compute a linear combination of an object's characteristics to determine its class. The characteristics are also called feature values. As an example, if one classifies images, the feature values can be the pixel values.

We speak about binary or binomial classification when there are only two classes. In this case the decision is made with both a threshold and the inner-product between object features and linear classifier coefficients.

When there are more than two classes there are two possibilities:

- *One-vs.-rest*, in which a binary classifier is built for each class in order to distinguish between this class and all the others. The decision is made as a function of the resulting dot product amplitude.

- *One-vs.-all*, in which a binary classifier is built for any pair of classes. The decision is made as a function of the number of positive votes received by each class.

3.2.2 Extremely randomized trees classifier

In ensemble learning methods, the predictions of several (usually small) base classifiers are combined in order to make an aggregated classifier which is more powerful and more robust than separate ones [Die00]. One of the possibilities to build an ensemble method is to average the decisions of many base classifiers. The combined classifier is stronger than any of the base classifiers.

A decision tree classifier [SL91] represents a tree-like structure where an internal node is a test on a single data feature, node output edges are the outcomes of this test and the tree leafs are decision classes. A decision tree classifier prediction is built by following a tree path from the root node to a leaf node. At each step a decision is made as a function of node condition.

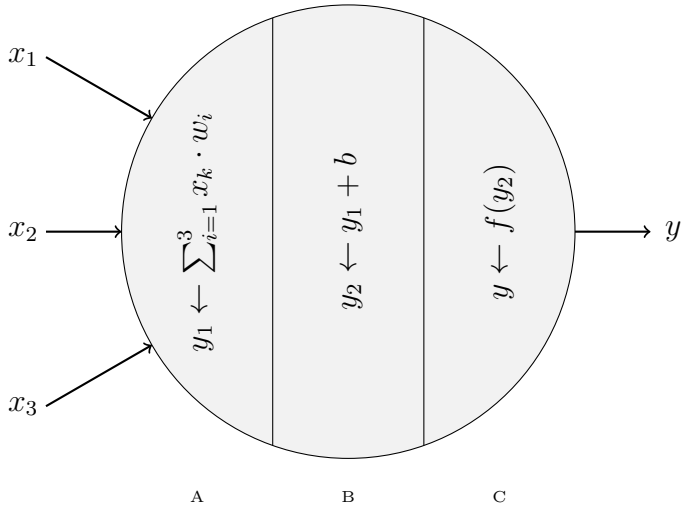


Figure 3.2: An artificial neuron with its 3 internal steps: (A) computes the weighted sum, (B) adds the bias, and finaly, (C) computes the activation function f. x_1, x_2 and x_3 are its inputs and y is the value it outputs.

Extremely randomized trees (ERT) classifier is an ensemble learning method in which base classifiers are decision trees. Roughly speaking, an ERT classifier builds many decision trees on different sub-sets of input data features. Prediction is performed by averaging the classes resulting from each decision tree. For more details, please refer to [GEW06].

3.3 Artificial neural networks

Artificial Neural Networks (ANN) [Yeg09] are a computational model inspired by the biological brain, included in the supervised machine learning methods. They are built with artificial neurons, a high level abstraction of real neurons. An artificial neuron is a function that firstly computes a weighted sum between its input values and predefined weights, then it adds a bias, and finally, the obtained value is passed through a so called activation function which can be among others the sigmoid function. The activation function's output is the neuron output. Figure 3.2 shows in detail an artificial neuron.

Artificial neural networks are organized in layers of neurons, outputs of one layer's neurons are the next layer neuron's inputs. Figure 3.3 shows an example of such architecture.

Neural networks include a lot of different algorithms as perceptrons, Hopfield networks, Boltzmann machines, fully connected neural networks,

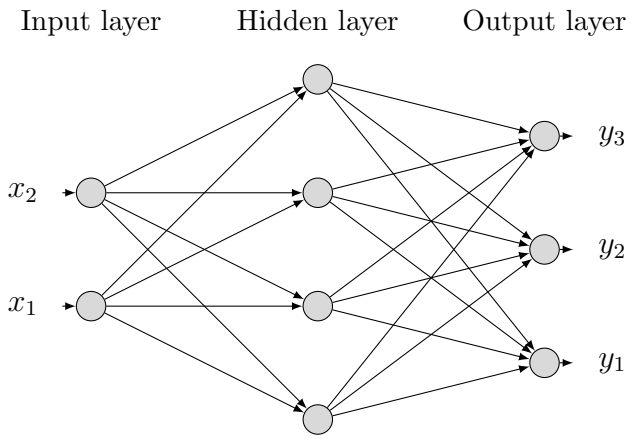


Figure 3.3: A neural network architecture.

convolutional neural networks, recurrent neural networks, long short term memory neural networks, autoencoders, deep belief networks, or generative adversarial networks among others.

3.3.1 Fully connected neural network

In a *fully connected neural network* each neuron is connected to every neurons in the previous layer. Figure 3.3 is in fact a fully connected network.

It used to be the dominating algorithm in neural networks. It is still the first type that comes in mind when talking about neural networks. Note that they have a huge number of parameters because each connection has it's own weight. It makes them quite expensive in terms of computation and memory.

3.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) [LBBH98, MMCS11] are a specific type of ANN inspired by the organization of the animal visual cortex, and have demonstrated excellent performance for computer vision and natural language processing.

In a convolutional neural network, each neuron is only connected to a few neurons from the previous layer and they also share weights. It considerably reduces the number of parameters in comparison with a fully connected network and make it relatively cheap in terms of computation and memory.

There are three main types of layers to build a convolutional neural network: *convolutional layer*, *pooling layer*, and fully connected layer.

3.3.3 Generative Adversarial Networks

In a generative adversarial networks [GPAM+14] there are 2 players represented by neural networks and one of them (the generator) is trying to forge new elements that look like those in a certain distribution while the other neural network (the evaluator) tries to determine whether this forged element is from the distribution or not. The GAN is trained is a way that when the evaluator detects a fake element, the generator tries to adapt to the feedback send a new forged element again and again.

3.3.4 Activation functions

In an artificial neural network, every neuron has an activation function. It can be for example the rectified activation function, the sigmoid function, or the tanh function among others. It comes from the biological model of neural networks where neurons fire an action potential or not (are activated or not), if the sum of its inputs signals exceeds a given threshold.

An artificial neuron calculates a weighted sum of its input and adds to it a bias and finally applies its activation function to that value. This result is the output of the neuron. Because it exists different activation functions with different properties and effects, when one is designing a neural network model, he as to chose carefully those functions.

The *step function* that outputs 0 if the sum is less than the threshold and output 1 otherwise, if the closest activation function to the biological neuron behavior. The fact that it is a binary function makes that it can work for a binary classifier, but what if the classifier has more than two classes? We are unable to determine the class when we have more than one neuron activated. This is why we need not binary activation function.

We will give details about two function: the rectified activation and the sigmoid function.

Rectified linear unit

A rectified linear unit (ReLU) is an artificial neuron employing the *rectifier activation* function which is defined as the function $f(x) = max(0, x)$. It outputs the input if it is positive and 0 otherwise. Figure 3.4 shows its graph. It was introduced in 2000 [HSM+00] and is now a very popular activation function for deep neural networks.

It is a non linear function and a helpful one because any function can be approximated with a combination of ReLU. Nevertheless it is not bounded, which means that several layers of ReLU can blow up. Because it is a simple

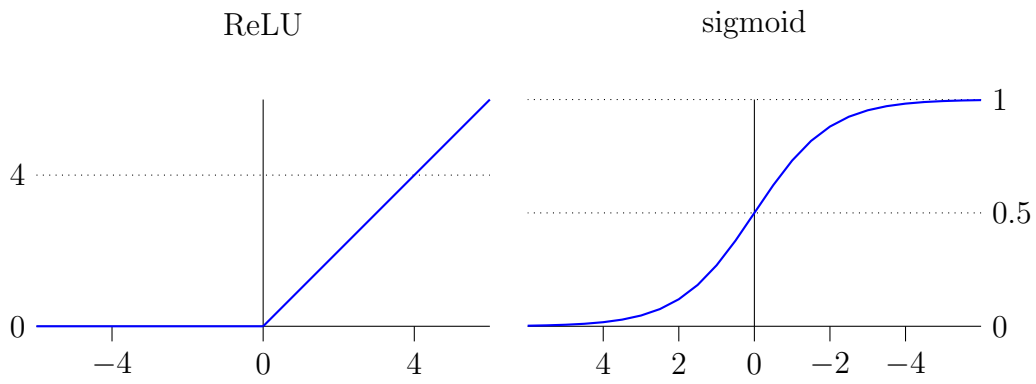


Figure 3.4: Two activation functions: ReLU and Sigmoid.

mathematical operation, ReLU is less computationally expensive which is a great advantage when one is designing deep neural networks. Note that ReLU can cause some neurons to die, meaning that they output 0 and make a part of the network passive. This is called the dying ReLU problem and is the origin of some variations of ReLU as "leaky ReLU" that prevent this situation to happen.

Sigmoid activation function

The sigmoid function is one of the most widely used activation function. It is defined as the function $f(x) = \frac{1}{1+e^{-x}}$. Figure 3.4 shows its graph.

It is non linear and its combinations are also non linear which means that it is not a problem to have layers of neurons with sigmoid as their activation function. It is not a binary function: it has infinite Y values all between 0 and 1. Nevertheless this function has a tendency to bring the Y values to its bounds. In fact a small change in the X value when we are between -2 and 2 modifies the Y value significantly. The sigmoid function is therefore great for classification. Another good thing is that it prevents output values from blowing up, it stays between 0 and 1. Note that because of its two plateaux (the 0 plateau on the left and the 1 plateau on the right of the graph), a network can reach a point where it learns very slowly or is not learning anymore, but there are ways to work around this issue.

3.3.5 Neural network measures

The process of learning for a neural network is achieved in terms of adaptation of the network parameters. With a supervised network, those parameters

may be changed according to the error measurement between the output of the model and the desired output. When a network is learning, it actually tries to minimize this error measure, iteration after iteration. This minimization might be performed by gradient descent optimization method.

Note that we must avoid to train to much a model. If it is not stopped at the right time, it will overlearn, which means that the neural network replace the relevant information of the general case by information from the individual cases.

We will now give details about two error measurements that we will need in this thesis.

Mean squared error measure

The *mean squared error measure* (MSE) is the simplest and the most used error function in neural networks for regression.

The mean squared error measure for two vectors $\vec{v}$ and $\vec{w}$ is defined as $\frac{1}{n}\sum_{i=1}^{n}(v_i - w_i)^2$. MSE measure is always non-negative, and the best value is 0.

Cross entropy measure

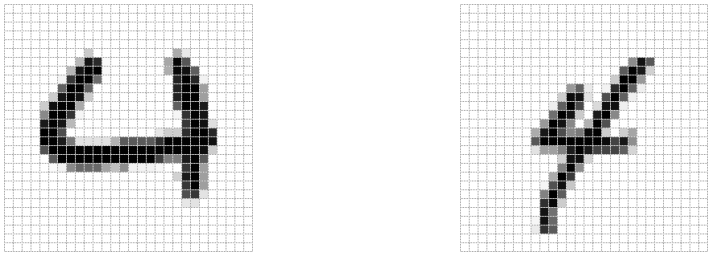
It is sometimes preferable to use the cross entropy measure instead of the MSE. Indeed it may accelerate the backpropagation algorithm. Sometimes, when one uses the MSE measure for training a neural network, the reduction of the error value can be extremely slow. The cross entropy can help avoiding those periods of stagnation [NBJ02].

This measure aim to quantify the difference between two probability distributions. Its formula is $H(p,q) = -\sum_x p(x)\log q(x)$ where $p(x)$ is the wanted probability and $q - x$ the actual probability.

3.4 Other related details

3.4.1 MNIST dataset

The MNIST dataset of handwritten digits [LCB] is a database composed of a training set, a test set and all the corresponding labels. It is a very famous and popular dataset in the machine learning community. Therefore it is a good set for performances comparison. Image sizes are $784 = 28 \times 28$ pixels and each pixel has 256 levels of gray. The digits have been size-normalized and centered in a fixed-size image. There are 10 possible labels (digits from 0 to 9). This database has been used a lot for the validation of



(a) (b)

Figure 3.5: Sample digit images from the MNIST database: (a) is the 60th image and (b) is the 61st.

classification algorithms. Refer to [BCD+94, LBBH98] for more information about machine learning studies over this dataset.

Figure 3.5 shows two samples from the MNIST dataset with a grey leveled representation. In this thesis we prefer to show MNIST images and other similar kind of images with a colored representation of the levels of grey.

3.4.2 Principal component analysis

Principal Component Analysis (PCA) [WEG87, Jol86] is a mathematical algorithm that reduces the dimentionality of the data while retaining most of the data set information. It identifies linear combinations of the original variables containing most of the information and basically goes from correlated variables into a smaller set of uncorrelated variables called principal components. Then using just those few principal components, each sample can be correctly represented with less variables.

Since it is just linear combinations that are needed, the PCA procedure simply outputs a matrix called *correlation matrix*, and the transformation is done by computing a matrix product between the correlation matrix and an input sample.

It is often used as a preprocessing for reducing the dimentionality of a machine learning dataset.

Part II

Contributions

Chapter 4

Privacy-preserving classification based on functional encryption

Privacy preserving computation allows multiple parties to run a function over their private inputs while the output is revealed.

With the generalization of data outsourcing, more and more concerns raise about the privacy and the security of outsourced data. In this context, machine learning methods have to be conceived and deployed, but with users privacy concerns addressed.

In a privacy preserving data classification process, one has to be able to extract knowledge (e.g. in the case of a classifier, deduction of the class label of an individual without compromising his private data) by assuring the protection of the sensitive data and, if possible, by hiding data access patterns from which useful properties could be inferred.

This is why privacy preserving classification [MWF08, YZW05] is a complex challenge. Several approaches have been proposed, using perturbation techniques like randomization [AH05, YZW05], condensation [AP04], or using k-anonymization [FWP07].

These approaches are different from the ones using Fuly Homomorphic Encryption (FHE), as for example [CdWM+17]. Indeed, with FHE one delegates the computation of the classification algorithm to a server and gets an encryption of the result, while with FE the computation is also delegated to the server but the server gets the unencrypted result of the classification too. Hence, with FE the server is able to perform the classification and get the output while with FHE it cannot. They are really different settings aiming at very different use cases.

The goal of a privacy preserving classification system is, as an example,

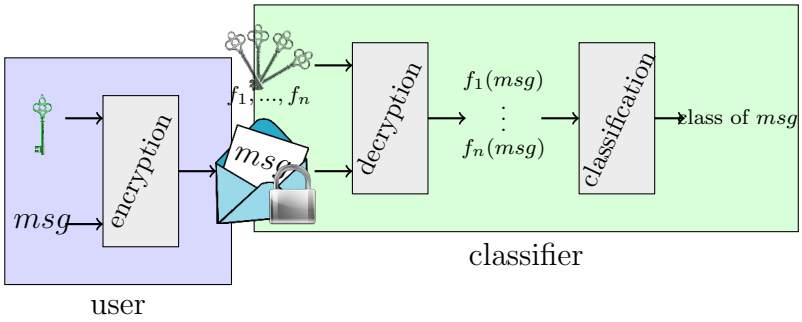


Figure 4.1: Privacy preserving classification using functional encryption. The classifier uses n secret keys. Each one enables an evaluation of a function from $\{f_i\}_{1 \leq i \leq n}$.

to deal with medical data that sometimes have to observe restrictive laws concerning their open access. It would be useful in this context (or for every kind of private data) and would enable to perform statistical studies without having a full access to the data.

In this chapter we describe our privacy-preserving classification algorithm based on functional encryption. We also give some details about our implementation of an inner-product functional encryption scheme [ALS16] that we used for experimentation. Since our algorithm requires a classification algorithm, we describe three different classification algorithms that we experimented. Finally, we give the classification performances obtained over the MNIST dataset [LCB] in different contexts.

4.1 Overview

As mentioned before, privacy-preserving classification is a complex challenge that has many different possible approaches. In this thesis, we propose a privacy-preserving classification algorithm based on functional encryption. The idea is to partially decrypt ciphertexts using a functional encryption scheme, and then to classify those outputs, consequently, original plaintexts are not revealed and classification is indeed performed. Figure 4.1 illustrates this concept. The privacy remains due to the functional encryption scheme. Roughly speaking, the data item on which a prediction must be made is encrypted. From the encrypted data, information is extracted and is used afterwards to produce the class of the data.

In our protocol, there is an entity called the *server* that has already performed the training step of a classification process. The output of this training process is used to determine the coefficients of the functionalities

$f_1, ..., f_n$ that will be associated with the n secret keys (as shown in Figure 4.1). The server wants to keep its classifier weights secret, but he also wants to classify data with it, without delegating its computation.

It may be possible to have many *users* which have data that they want to keep secret but, at the same time, they also want to release the classification results of those data to the server (for example in order to obtain a service). There is a third party that both the server and the users can trust, named *authority*, that will be the functional encryption scheme authority. The authority only plays a role when the privacy-preserving classification system is set up, neither the server nor the users need the authority to encrypt or classify once the keys (public key and secret keys) are all generated. Its goal is in a first step to check that the server's functionalities $f_1, ..., f_n$ are not dishonest (the server is not trying to rebuilt the original plaintexts). In a second time, the authority has to generate an instance of an inner-product encryption scheme.

We now describe in detail this protocol, illustrated in Figure 4.2. The initialization phase has two steps:

First, the authority generates the public key and the master secret key with the `Setup` algorithm of the FE and publish the public key.

The following steps are repeated each time a server wants to join the system:

(A) The server uses the training algorithm on his training set.

(B) The authority receives the $f_1, ..., f_n$ from the server, checks them and generates the private keys $\{sk_{f_i}\}_{1 \leq i \leq n}$ using the `Keygen` algorithm of the FE, and afterwards sends them to the server.

The following steps are repeated each time a user sends its data to a server:

(1) The user encrypts private data msg with the `Encrypt` algorithm of the FE, and sends it to the server.

(2) The server decrypts it with all of his secret keys $\{sk_{f_i}\}_{1 \leq i \leq n}$ using the `Decrypt` algorithm of the FE, and obtains the evaluations $\{f_i(m)\}_{1 \leq i \leq n}$ of FE scheme's functionalities.

(3) The server uses its classification algorithm in order to predict the class of private data m using values $\{f_i(msg)\}_{1 \leq i \leq n}$.

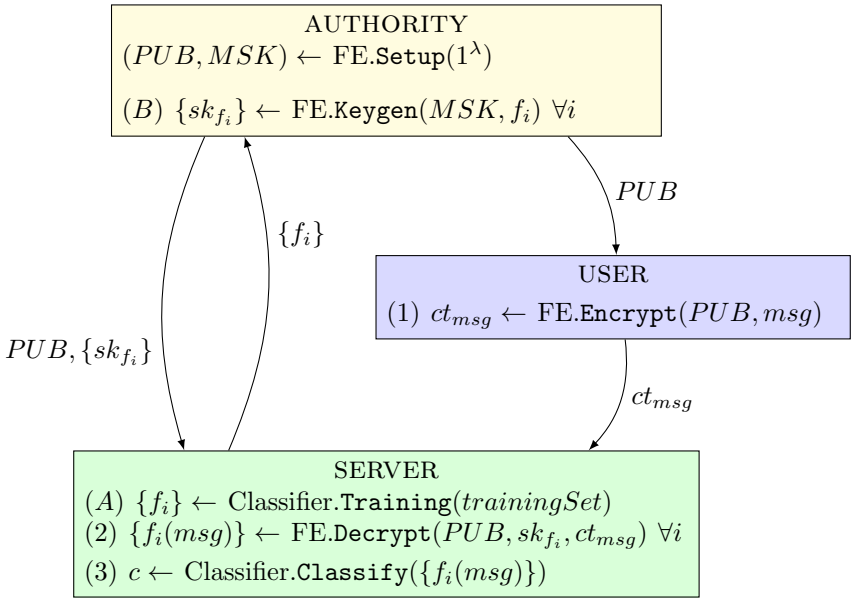


Figure 4.2: Privacy preserving classification protocol using a functional encryption system.

We experimented this construction using inner-product functional encryption [ALS16] over the well known MNIST database [LCB]. Details about our IPFE implementation are given in 4.2.

To illustrate the advantage of our construction, we now describe a realistic use case. We assume that there is a pharmaceutical company which has constructed a classifier that takes as input medical and private pieces of information. The company also does not want to divulge its secret classifier. However, it wants to conduct a study on real human data, for example the patients of a hospital. The law about patient medical data disclosure can be very restrictive depending on the country. For instance in France, a hospital cannot share private data of its patients without a slow and cumbersome administrative process. Our construction provides a solution to this use case. We simply need a third party that can be trusted by the company and by the hospital. For example, a governmental agency should be able to do it. So, the agency generates the public key and the secret keys associated with the company classifier. The hospital encrypts the data of its patients and sends it to the company. The company decrypts them with its secret keys. After decryption, the company only gets the result of a computation which involves the patients data. It is important to notice that the critical data remains encrypted during the whole process. Nevertheless, the company can use the computation result (i.e. the inner-products) to perform the classification.

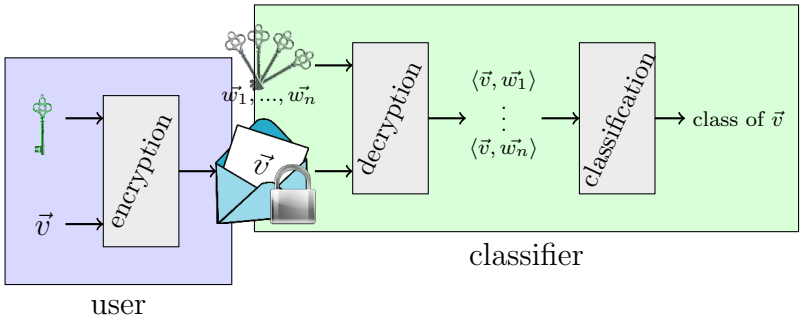


Figure 4.3: Privacy preserving classification using functional encryption. The classifier uses n secret keys. Each one enables an evaluation of an inner-product with a vector from $\vec{w_1}$ to $\vec{w_n}$.

Proceeding in this way, it can perform its study on real medical data without endangering patient data privacy. Moreover, our solution can involve several hospitals and companies if needed (the agency has to generate new secret keys for each new classifier).

It is also important to notice that there is always a risk of collusion. In the previous scenario, it means that few classifiers can share their secret keys and extract more information than what they are allowed to have. The effective security of our construction is studied in the next chapters.

4.2 Implementation of inner-product functional encryption schemes

Agrawal et al. proposed [ALS16] two different constructions of fully secured inner-product functional encryption schemes. One has its security based on the learning with errors (LWE) problem, and the other one security is based on the decisional Diffie-Hellman (DDH) problem. The recommendations from the French Network and Information Security Agency (ANSSI) [ans] suggest to use, for the latter, 2048 bit groups in a discrete logarithm context.

We implemented the DDH construction following this recommendation. Which means using a prime field $\mathbb{F}_p$ such that p is a safe prime of approximately 2048 bits. The description of the context and the algorithms are given in section 2.3. The group where DDH is assumed to be difficult is the subgroup of $\mathbb{F}_p^*$ which has the prime order size of $(p-1)/2$, we called it $\mathbb{G}$. Refer to Section 2.1.1 for more details about DDH assumption.

The decryption of the IPFE scheme is performed by computing a discrete logarithm. In our case, the inner products obtained by the application of

plaintext	ciphertext	secret key
784 B	196 kB	1296 B

Table 4.1: Sizes from our IPFE implementation. The secret key also counts the coefficients of the vector that the secret key is associated with. Plaintexts are vectors of size 784.

the learned linear coefficients on the MNIST database belong to a small interval. We have pre-computed a lookup table with all the (α, g^α) for α in $\{-933197 \dots 424769\}$. This pre-computation took 6.2 seconds and the size of the obtained lookup table is about 337MB. The size of the lookup can be reduced to 10.4MB if only the first 32-bits of all the g^α are kept, which can be seen as a hashing of g^α (less than 0.1% of collisions are obtained in this case). Solving the discrete logarithm using the lookup table takes under 0.18 seconds. We measured the execution time of the algorithms: 0.0004 seconds for `Keygen`, 0.15 seconds for `Encrypt` and 2.3 seconds for `Decrypt`.

The IPFE algorithms were implemented in C++ using the FLINT library [HJP13] for computations over field $\mathbb{F}_p^*$ computations. The experiments were performed on a regular laptop computer with an Intel Core i7-4650U CPU and 8GB of RAM. A ciphertext in 256 times bigger than a plaintext and a secret key has a size of 1.3 kB, the exact sizes of the elements we manipulate in the IPFE instantiation are given in the table 4.1. Those sizes are given in the context of a classification algorithm applied to the MNIST database: the plaintexts are an images (vectors of size 784), the ciphertexts are encrypted images.

We have an implementation that is practical both in terms of space in time. Indeed, the step that takes the most time is the decryption process, and it needs less than 3 seconds without any optimization on a regular laptop. Details about the execution times are presented in Table 4.2. The key generation process and the encryption process are faster. The key generation and the decryption process can be parallelized: for any number of secret keys, the computation time can be reduced to the time needed for just one key generation and decryption.

4.3 Privacy-preserving classification based on inner-product functional encryption

For practical purpose, we used an inner product functional encryption scheme [ALS16] to experiment our construction. Details about our implementation

Number of secret keys	Keys generation(s)	Encryption	Decryption(s)
1	0.0004 s	0.15 s	2.3 s
10	0.004 s		23 s
20	0.008 s		46 s
30	0.012 s		69 s

Table 4.2: Execution times (in seconds) for the inner product functional encryption implementation. The key generation and decryption columns contain the needed time without any parallelization. Secret keys are associated with vectors of size 784.

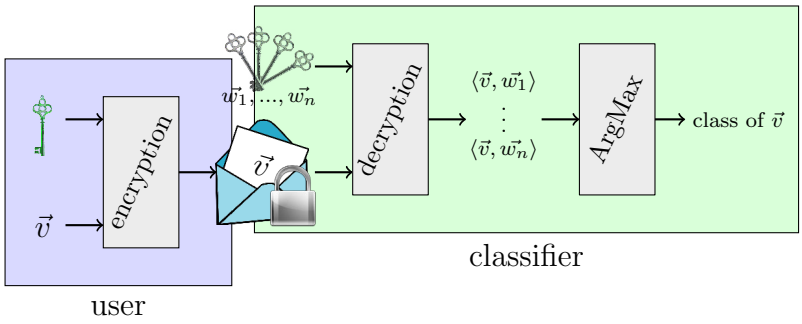


Figure 4.4: Privacy preserving classification using functional encryption with a linear classifier. The classifier uses n secret keys. Each one enables an evaluation of an inner-product with a vector from $\vec{w_1}$ to $\vec{w_n}$. There are $m = n$ classes with this classifier.

are given in 4.2.

The use of an inner-product functional encryption implies that messages are basically vectors. The secret keys are associated with vectors as well, and after decryption, we get an inner-product between a plaintext and secret key's vector.

Figure 4.3 shows the inner product functional encryption specification of our construction. As we can see, the privacy-preserving classifier has n secret keys, previously provided by the authority (not mentioned in this figure). As soon as it gets a ciphertext, it decrypts the ciphertext n times with n secret keys. Then, there is the machine learning part: those inner-products have to be used by a ML algorithm to classify the encrypted vector $\vec{v}$. This is the object of this section that describe 3 algorithms that we used: linear classifier, extremely randomized tree classifier and, finally, neural network classifier. Details about those algorithms can be found in 3.2 and 3.3.

4.3.1 Linear classification

The simplest way to classify is to use a linear one-vs.-rest classifier (refer to Section 3.2.1 for details about linear classification). Figure 4.4 shows this construction. Since it is a linear classifier, we have $m = n$ classes, each one represented by vectors from $\vec{w_1}$ to $\vec{w_n}$.

The training step outputs the vector set $\{\vec{w_i}\}_{1 \leq i \leq n}$. These are the vectors associated with the secret keys that will be generated by the authority and then given to the privacy preserved classifier. We suppose that a vector $\vec{v}$ has been encrypted into a ciphertext by an user. When the privacy-preserving classifier gets it, it will decrypt it n times with the n secret keys. Then, it will just have to compute the $argMax(\{\langle \vec{v}, \vec{w_i} \rangle\}_{1 \leq i \leq n})$ which gives the class of the vectors $\vec{w_1}$ to $\vec{w_n}$.

The linear classifier is not the best in terms of classification performance, which is why we tried two other algorithms.

4.3.2 Extremely randomized trees classification

In order to improve our privacy-preserving classifier, we have replaced the linear classifier with an extremely randomized tree classifier. For details about this type of classifier please refer to Section 3.2.2. Figure 4.5 shows the updated construction of our scheme. We keep the same secret keys associated with the same vectors. This time, the privacy-preserving classifier has to train an ERT classifier over a transformation of the original training set. In fact this time instead of learning with a vector $\vec{v}$ from the training set, we will learn with $\{\langle \vec{v}, \vec{w_1} \rangle, ..., \langle \vec{v}, \vec{w_n} \rangle\}$.

Then, when using the system of privacy-preserving classification, in the same manner, inner-product values obtained after decryption are provided to the ERT classifier. The ERT classifier succeeds in extracting information from the n inner-products values and classify with better performances that the linear classifier.

Using the extremely randomized tree classifier improves the performances but we can still make it even better.

4.3.3 Neural network classification

We now reach our best solution in terms of performance of classification. The ERT classifier is now replaced with a neural network classifier. For details about neural networks please refer to Section 3.3.2. Figure 4.6 shows the updated construction.

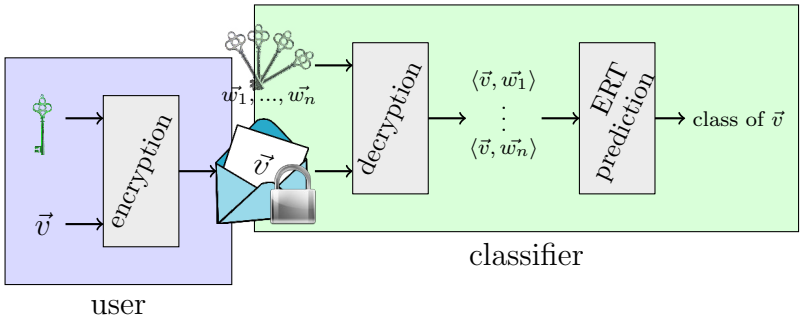


Figure 4.5: Privacy preserving classification using functional encryption and ERT classification. The classifier uses n secret keys. Each one enables an evaluation of an inner-product with a vector from $\vec{w_1}$ to $\vec{w_n}$.

The classifier party trains a neural network model with the training set from Figure 4.7. The first layer of this network model reproduce the inner-product from the IPFE's decryption. So it is composed by n neurons for the n secret keys of the system. As soon as the model is trained, the n vectors have been generated in the first layer of the model. The authority can generate the n secret keys associated with those vectors.

The privacy-preserving classifier party has now to remove the first layer of the neural network (which becomes the network from Figure 4.8), and to train it a little more with (as done with the ERT) this time not with the training set but instead with the inner-products of the training set vectors. This has to be done because values are converted from integers (inner products) to reals between 0 and 1 (network inputs) which cause a little decrease of the classification efficiency. Finally, once a ciphertext is decrypted n times with the n secret keys, the classifier party uses its neural network and gets the class of the plaintext.

The fist layer of the model from Figure 4.7 is a fully connected layer with the identity function as the activation function. The rest of this model is the final neural network model (Figure 4.8), and it is composed of 3 hidden layers. Every layers are fully connected and all the activation functions are the ReLU function, except for the last one which is the sigmoid function.

In this context, the number of classes m can be different from the number of secret keys n. n determines the size of the input layer of the neural network model and m the size of the output layer of the model.

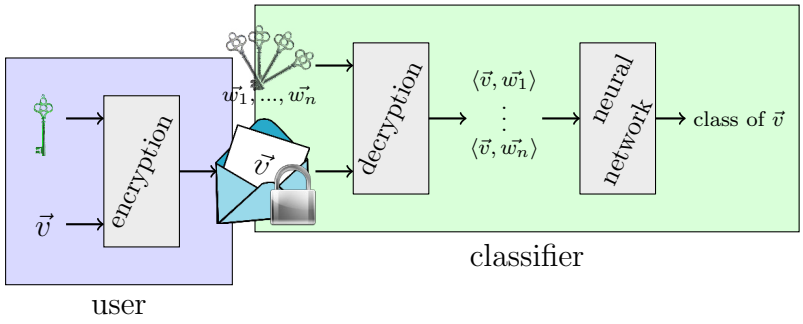


Figure 4.6: Privacy preserving classification using functional encryption and a neural network classifier. The classifier uses n secret keys. Each one enables an evaluation of an inner-product with a vector from $\vec{w_1}$ to $\vec{w_n}$. There are $m = n$ classes with this classifier.

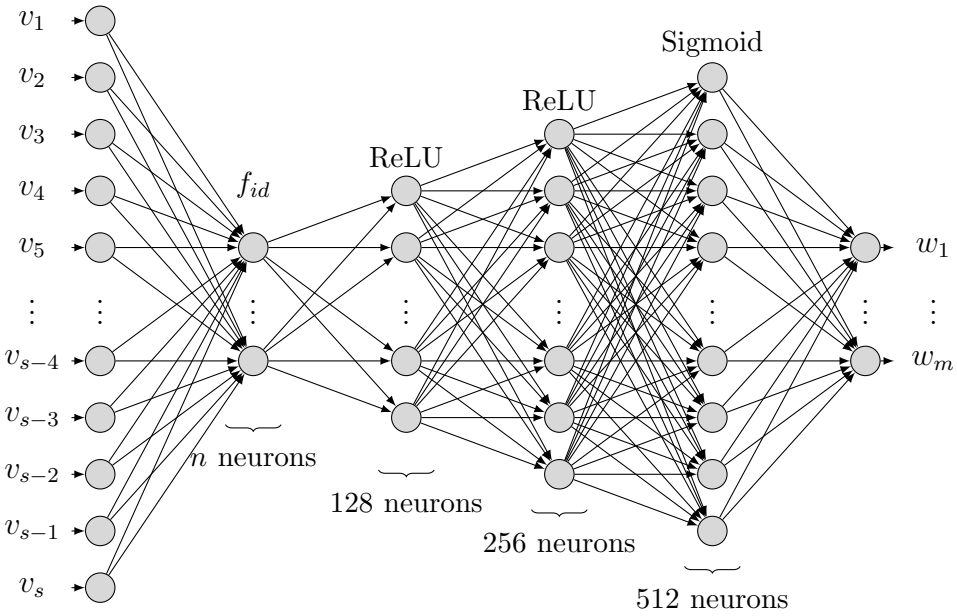


Figure 4.7: Description of the ANN model used to generate the vectors for secret keys. All the layers are fully connected layers. Variable n stands for the number of secret keys, and m for the number of classes (or subclasses) according to the classifier. Variable s denotes the length of the vectors from the dataset, and also the length of vectors $\{w_i\}_{1 \leq i \leq n}$.

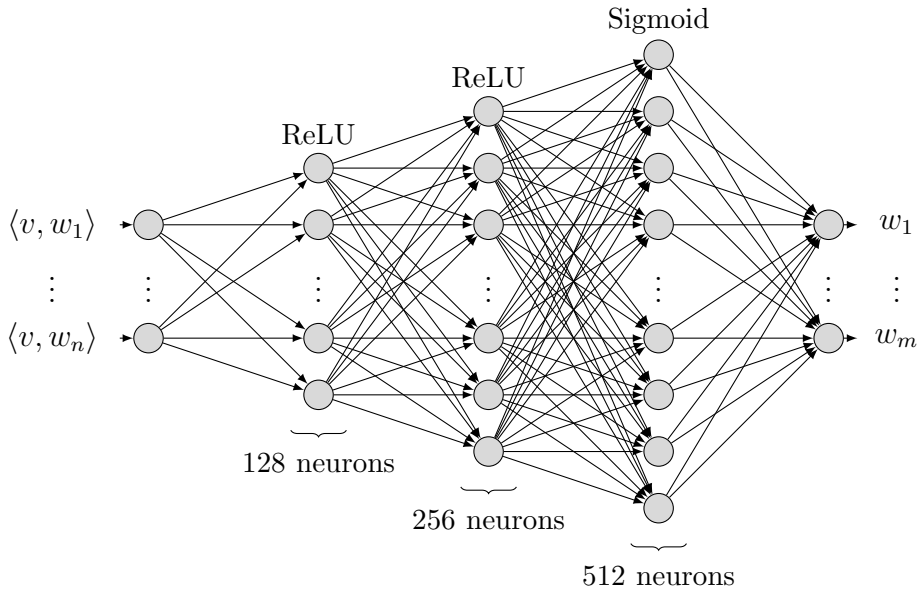


Figure 4.8: Description of the ANN model used for the classification. All the layers are fully connected layers. Variable n standn for the number of secret keys, and m for the number of classes (or subclasses) according to the classifier.

Classifier	Learning	Prediction	Performance
Linear	6 sec.	< 0.1 sec.	86.07%
ERT	30 sec.		92.68%
NN	185 sec.		96.69%

Table 4.3: Execution results of the proposed classifiers in a $n = 10$ secret keys scenario.

4.3.4 Experimental results

We experimented the three privacy-preserving classifiers we have described above on the MNIST dataset of handwritten digits [LCB]. Please refer to Section 3.4.1 for more details. Since we use the MNIST dataset, m has a fixed value of 10 classes.

The experiments were performed on a regular laptop computer with an Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz and 16GB of RAM. Classifier error rate is the percentage of miss-predictions reported to the total number of predictions.

Classifier	Performance
ERT with $n = 10$ classes	92.68%
ERT with $n = 20$ classes	95.14%
ERT with $n = 30$ classes	95, 64%

Figure 4.9: Error rates for the ERT classifier.

Linear classification

Table 4.3 presents execution times for the 3 proposed classification methods when we use $n = 10$ secret keys. The performance of the linear classifier is about 86%.

It is possible to increase the performance of these privacy-preserving linear classifiers. Indeed, we can further split each input class into several sub-classes. This corresponds to assigning new "artificial" labels to input data set (simply relabeling the target values).

ERT classification

As we can see in table 4.3, the ERT classifier is better than the linear one. Indeed, using the extremely randomized tree learning process enables to increase classification performances. Its performance is about 93% with 10 secret keys.

It is also possible to increase the performance of this privacy-preserving ERT classifiers. In the same way than with the linear one, we can further split each input class into several sub-classes by simply relabeling the target values. This corresponds to assigning new "artificial" labels to the input data set. We give experimental results of that in table 4.9, with $n = 20$ and $n = 30$ secret keys.

Neural network classification

As we can see in table 4.3, the neural network classifier is better than the linear and the ERT ones. Its performance is about 96%.

Details about neural network models used are given in Figures 4.7 and 4.8. In our experimentation, the numbers of neurons in the hidden layers are sequentially 128, 256, and finally 512. The number of classes is $m = 10$ because of the 10 digits from the MNIST dataset. The variable n represents the number of secret keys and is set to 10 at first for comparison with the previous results.

We also experimented values for n varying from 1 to 9. Classification performances of those experiments are given in Figure 4.10. We emphasize

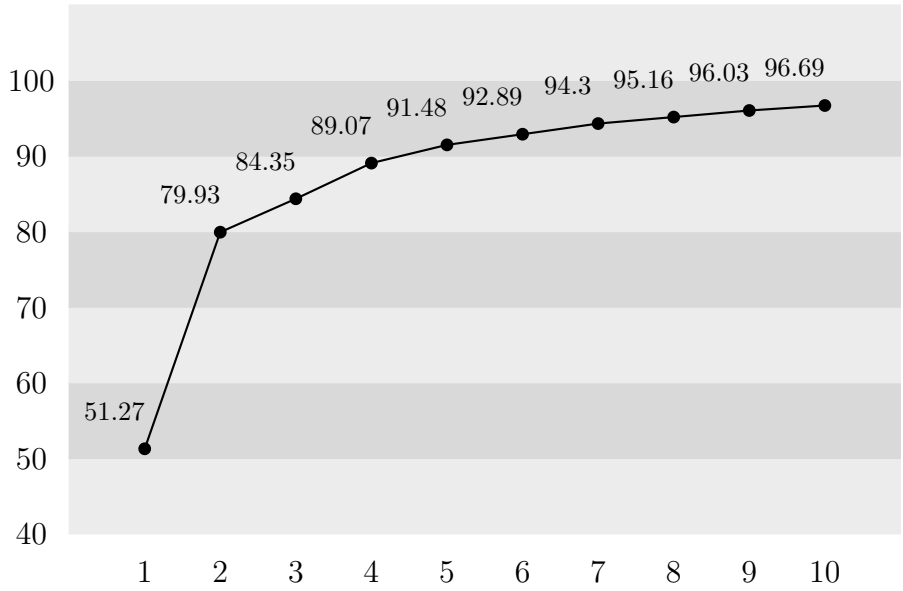


Figure 4.10: Classification performance of the inner-product functional encryption based privacy-preserving classifier. The x-axis shows the number of secret keys n varying (from 1 to 10), and the y-axis shows the percentage of classification success.

the fact that the use of a functional encryption scheme does not really change anything from the classification point of view. Indeed it just force us to classify projections (inner-products) of input (images) instead. Note that best classification performances on the MNIST dataset reach more than 99% [WZZ+13]. With only $n = 6$ secret keys, the performance of the classifier reaches 93% which is better than what we can do with $n = 10$ secret keys and the ERT classifier.

The previous classification performances are average values over all digits. However, we notice that there is a disparity between digits. Indeed, from one to another, the average classification performance can vary a lot: from 63% for the 8 to 94% for the 1 digit (with the $n = 2$ secret keys scenario). We show those disparities for the $n = 1$, $n = 2$ and $n = 10$ secret keys scenarios, in Figure 4.11. It suggests that some digits are harder to recognize than then others. It can be explained by the fact that some digits have a more complex shape, with curves for instance. The one which is a simple straight line is the easiest one to classify.

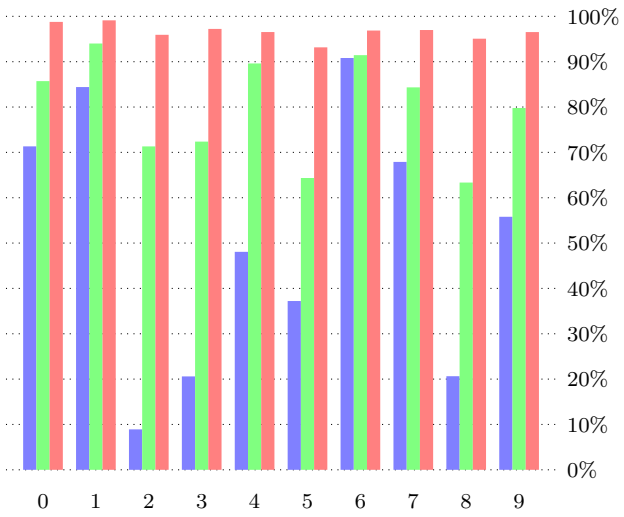


Figure 4.11: Classification performance of the inner-product functional encryption based privacy preserving classifier. The x-axis shows the 10 digits, and the y-axis shows the percentage of classification performance per digit. The blue, green and red bars respectively correspond to $n = 1$, $n = 2$ and $n = 10$ secret keys scenarios.

4.4 Privacy-preserving classification based on quadratic polynomial functional encryption

Functional encryption for quadratic polynomial functionality [BCFG17] is an attractive solution to increase the performance of classification or to reduce the number of secret keys to generate. We provide details about such schemes in Section 2.4. At the beginning of the thesis, there was no such construction in the literature, so we abstracted our privacy-preserving classifier from a practical quadratic polynomial functional encryption scheme. We indeed experimented our privacy-preserving classifier while considering that it was using a functional encryption for quadratic polynomial function instead of the inner-product functional encryption one. Figure 4.12 shows this different instance of our PPC construction. The advantage is that we can compute more precise prediction with less secret keys.

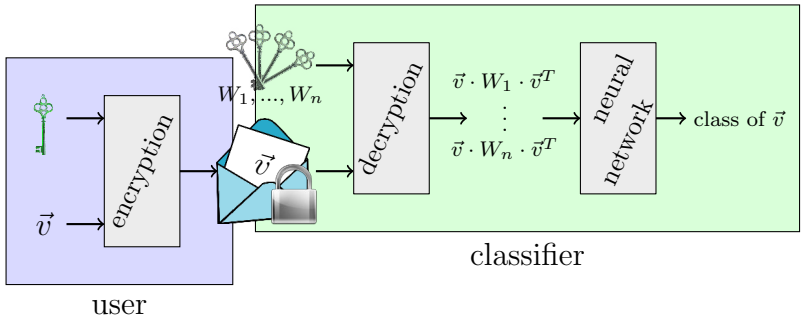


Figure 4.12: Privacy preserving classification using a quadratic polynomial functional encryption and a neural network classifier. The classifier uses n secret keys. Each one is associated with a matrix from W_1 to W_n.

4.4.1 Experimental results

In order to compare between quadratic polynomial and inner-product, we select only 784 monomials. So we have in both cases 784 constants.

We trained a classifier that does the quadratic polynomial step before being fully connected. Then we selected the 784 biggest weights from the quadratic polynomial layer.

In Figure 4.13, we give the classification performances of this construction still using the MNIST dataset of handwritten digits [LCB]. As we did in Figure 4.10, we give performances for scenarios with $n = 1$ secret key to $n = 10$. We also plot in this Figure the performances of the IPFE based classifier in gray (from Figure 4.10). As we can see the classification performances are better way better than with the IPFE based classifier. In fact, We have the same performances for $n = 5$ secret keys with the IPFE version than with only $n = 2$ keys with the quadratic polynomial functional encryption: 91%.

4.5 Discussion

Our construction combine functional encryption and machine learning to achieve privacy preserving classification.

We experimented mainly with an inner-product functional encryption scheme. We tried three different machine learning algorithms with it, and find out that the neural network was the best in terms of both classification performance and the number of secret keys needed. Indeed, with only five secret keys the performance is above 91%.

We also tried with a functional encryption schemes for quadratic polynomial combined with a neural network. It increased the performances and

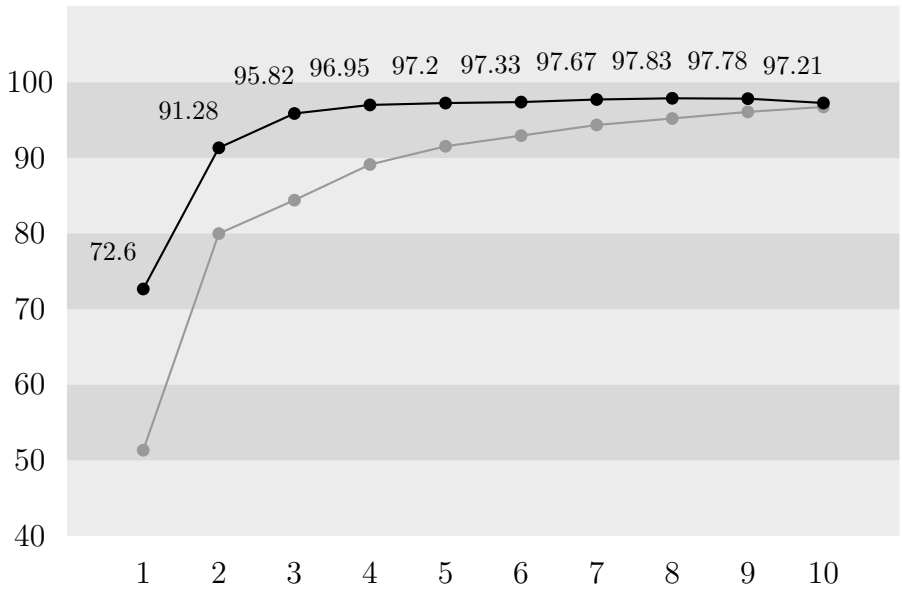


Figure 4.13: Classification performance of the quadratic polynomial functional encryption based privacy preserving classifier (black dots). The x-axis shows the number n of secret keys provided, and the y-axis shows the percentage of classification success. We add the gray dots corresponding to the performances of classification with an IPFE scheme (Figure 4.10) for comparison.

also reduced the number of keys needed.

Note that there are still few practical functional encryption schemes nowadays, but since the classification performances are independent of the functional encryption scheme used in our construction, it is possible to determine performances without having the practical FE scheme yet.

The functional encryption scheme that we used is fully secure under the decisional Diffie-Hellman assumption. Details about it are in 2.1.1. But since a functional encryption reveals after decryption only a part of the original plaintext, we want to know what can be known exactly in our privacy preserving classification use case. It is the object of the next chapter.

Chapter 5

Leakage-based attacks on functional encryption settings

In this chapter we study the functional encryption information leakage, in particular for the use case described in Chapter 4. We consider attackers who respect the protocol, but try to get more information than what they are supposed to. This investigation goes beyond the cryptographic security of the underlying FE scheme. Indeed, the result of the computation provided by a secure FE scheme may leak more information about the plaintext than expected. If, in a particular use case, this information leakage compromises the plaintext, it means the FE system may not be considered as secure for this particular use case, even if the underlying FE scheme is proved to be stand alone secure. Our goal is to study the security of a FE system when one has infomation about the plaintext's semantic, and one key or more. We propose attacks based on machine learning to estimate the information leakage, mainly focusing on inner product functional encryption. Finally, we experiment our attacks on the MNIST dataset [LCB] (details about this dataset are given in Section 3.4.1), and we provide experimental results.

5.1 Context of the attack

As a starting point of this security study we investigate our privacy preserving classification protocol based on functional encryption described in Chapter 4. Our attack setting is the following: the malicious classifier (the attacker) knows exactly the semantic of data that has been encrypted and has access to a training set (see Figure 5.1).

The owner of the secret key sk_f associated to the function f is malicious and designs an attack using only the output of the FE decryption algorithm

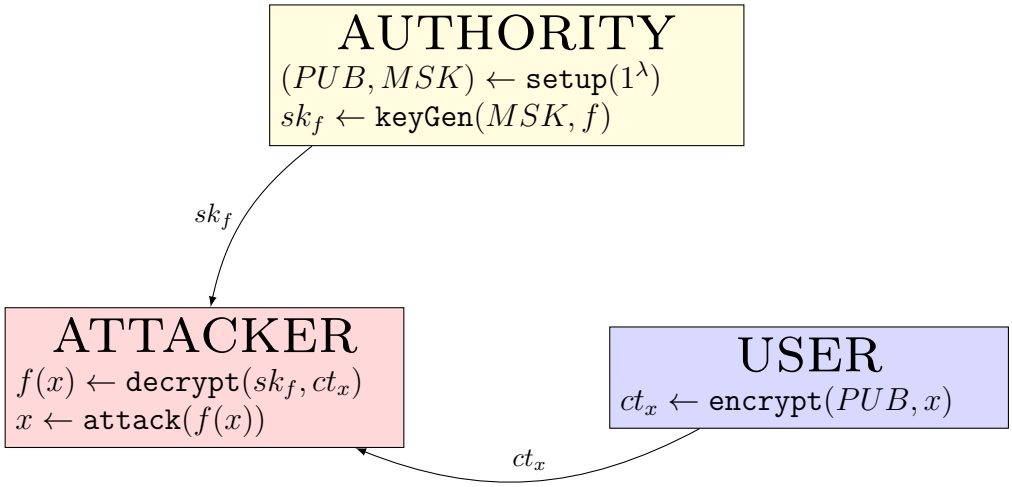


Figure 5.1: Description of the considered attack on a FE use case and the actors involved.

to get original plaintexts.

5.2 Attack methods

The attacker owns a set of n secret keys $\{sk_{\vec{w}_\ell}\}_{1\leq\ell\leq n}$ associated with the n vectors $\vec{w}_\ell$, $1 \leq \ell \leq n$. We consider vectors of length m. When he gets an encryption $ct_{\vec{v}}$ of a vector $\vec{v}$, he is able to compute $\{\langle\vec{v}, \vec{w}_\ell\rangle\}_{1\leq\ell\leq n}$ using the decryption algorithm of the IPFE scheme, which gives him the class c of $\vec{v}$ according to his own classifier.

So he has the following system with $\vec{v'}$ as unknowns.

$$\begin{cases} ip_1 &= \langle\vec{v'}, \vec{w_1}\rangle \\ ip_2 &= \langle\vec{v'}, \vec{w_2}\rangle \\ \vdots & \vdots \quad\quad \vdots \\ ip_n &= \langle\vec{v'}, \vec{w_n}\rangle \end{cases} \tag{5.1}$$

Solving this system is equivalent to finding all the vectors $\vec{v'}$ that satisfy the equation:

$$\vec{ip} = W \cdot \vec{v'} \tag{5.2}$$

with $\vec{ip}^T = (ip_1, \cdots, ip_m)$, $\vec{w'}^T = (w'_1, \cdots, w'_m)$ and the matrix W with n lines and m columns (composed by the n vertical vectors $w_1, ..., w_n$). The plaintext $\vec{v}$ is one of the solutions of the system and the difficulty to find

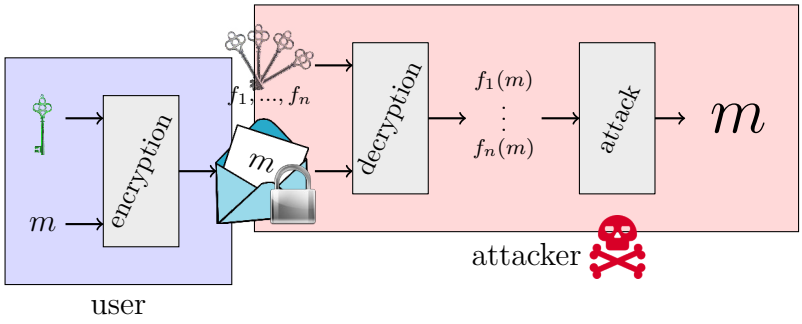


Figure 5.2: Description of the considered attack on a FE use case.

it depends on m, n, $\{\vec{v}^i\}_{1\leq i\leq n}$ and on the intrinsic properties of the used messages (specific images, particular data, random, ...).

Yet, in many cases, such an attack is possible. To illustrate it, we will consider the following example in $\mathbb{Z}$. Let $m = 16$ and $n = 4$. The plaintexts belong to $\{0,1\}^{16}$ which is included into the definition space $\mathbb{Z}^{16}$. Let $\vec{v}^T =$ (0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1). Vectors $\vec{w}_1 =$ (2, 81, 80, 82, 3, 78, 90, 14, 66, 29, 52, 36, 11, 40, 83, 31), $\vec{w}_2 =$ (70, 64, 65, 46, 74, 10, 2, 85, 23, 54, 2, 41, 95, 83, 38, 6), $\vec{w}_3 =$ (54, 43, 98, 0, 93, 78, 23, 91, 52, 39, 43, 62, 19, 57, 95, 50) and $\vec{w}_4 =$ (87, 49, 3, 33, 28, 47, 96, 18, 17, 8, 92, 69, 89, 38, 84, 10) are randomly chosen in $\{0, \cdots, 99\}^{16}$. So, we have $\vec{ip}^T =$ (460, 334, 502, 400). There are an infinite number of solutions in $\mathbb{Z}^m$ but only one in the subset that our plaintexts come from: $\vec{w} \in \{0,1\}^{16}$. With a brute force attack we find it in a few seconds even if $\vec{v}$ was encrypted with an inner product functional encryption scheme that is secure!

A small space is clearly insecure. That is why on the one hand the parameter m and the size of the "realistic" plaintext space has to be large enough, and on the other hand the number of inner products n has to be limited. Within our MNIST use case, a brute force attack is unthinkable because of the size of our plaintext space: $(2^8)^{784}$. Nevertheless it does not mean that it is not possible to get more information than the inner-product using vector $w_1, ..., w_n$.

There are k classes in his classifier : $1, ..., k$. The attacker's challenge is to recover input vector $\vec{v}$ from $\{\langle \vec{v}, \vec{w}_\ell \rangle\}_{1\leq \ell \leq n}$ and c. Because he determines the classes c of every plaintexts $\vec{v}$, he is able to specialize attacks for every class of its classification process. The attacker also knows what kind of data has been encrypted, so he is able to get a valid training set. He can then use it as he wants.

5.2.1 Principal component analysis

The idea of this attack is to use PCA properties to exploit the inner products values and recover an approximation of the original image. Refer to Section 3.4.2 for details about PCA.

Let $A \in \mathcal{M}_{n,m}(\mathbb{R})$ be the matrix composed of row vectors $\vec{w}_1, ..., \vec{w}_n$ and let $\vec{v}$ be a vector of c-class. Let $\vec{ip} = A \cdot \vec{v} = (ip_1, ..., ip_n)^T$ be the vector of all the inner products obtained with the FE scheme. So, for $1 \leq \ell \leq n$ we have $ip_\ell = \langle \vec{v}, \vec{w}_\ell \rangle$.

The attacker uses PCA for each class c over its training set. He gets k matrices : $\{M_{PCA,c}\}_{1 \leq c \leq k} \subset \mathcal{M}_{n,m}(\mathbb{R})$. The PCA procedure ensures the following property $\vec{v} \backsimeq M^T_{PCA,c} \cdot M_{PCA,c} \cdot \vec{v}$ when $\vec{v}$ is from the class c.

Then, the attacker computes matrices $\{P_c\}_{1 \leq c \leq k}$ such that for all $1 \leq c \leq k$, $M_{PCA,c} \backsimeq P_c \cdot A$. This can be done using a gradient descent algorithm (optimization algorithm used to find a local minimum for a continuous function). He determines the matrix P_c by minimizing $\sum_{i,j}((P_c \cdot A - M_{PCA,c})_{i,j})^2$.

He is now able to generate a vector relatively close to the original one. When he gets an encryption $ct_{\vec{v}}$ of a vector $\vec{v}$, he gets the $\vec{ip}$ vector with the IPFE decryption algorithm, then he determines its c-class and simply computes $M^T_{PCA,c} \cdot P_c \cdot \vec{ip}$:

$$\begin{aligned} M^T_{PCA,c} \cdot P_c \cdot \vec{ip} &= M^T_{PCA,c} \cdot P_c \cdot A \cdot \vec{v} \\ &\backsimeq M^T_{PCA,c} \cdot M_{PCA,c} \cdot \vec{v} \\ &\backsimeq \vec{v} \end{aligned} \tag{5.3}$$

In equation (5.3), we have the second line because $P_c \cdot A \backsimeq M_{PCA,c}$, and we have the third line because $\vec{v} \backsimeq M^T_{PCA,c} \cdot M_{PCA,c} \cdot \vec{v}$ when $\vec{v}$ is from the class c.

Figures 5.3 and 5.4 show some images obtained using the PCA attack over the MNIST dataset. We consider 3 instances of the attack here: 30, 20 and 10 secret keys scenario. As we can see on those figures, the more the attacker has secret keys, the better are the attack results. About Figure 5.4: the image in the first row is the one with the lowest MSE (2154) from the test set in the 10 secret keys scenario and the image in the second row is the one with the highest MSE (15590) from the the test set in the 10 secret keys scenario.

5.2.2 Fully connected network

This attack uses an Artificial Neural Network (refer to Section 3.3 for more details). The network model is composed of α hidden layers of fully connected neurons. Each hidden layer is composed of β_i neurons for $1 \leq i \leq \alpha$. The

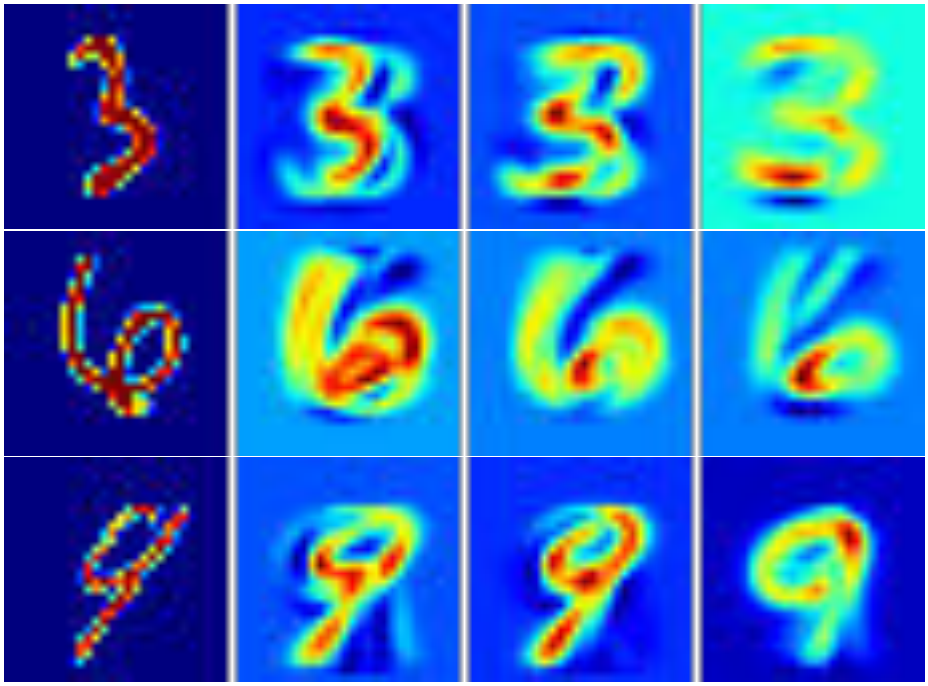


Figure 5.3: Samples of the PCA attack results. The columns are respectively (from left to right) original MNIST images, attack result in a 30, 20 and 10 secret keys scenario.

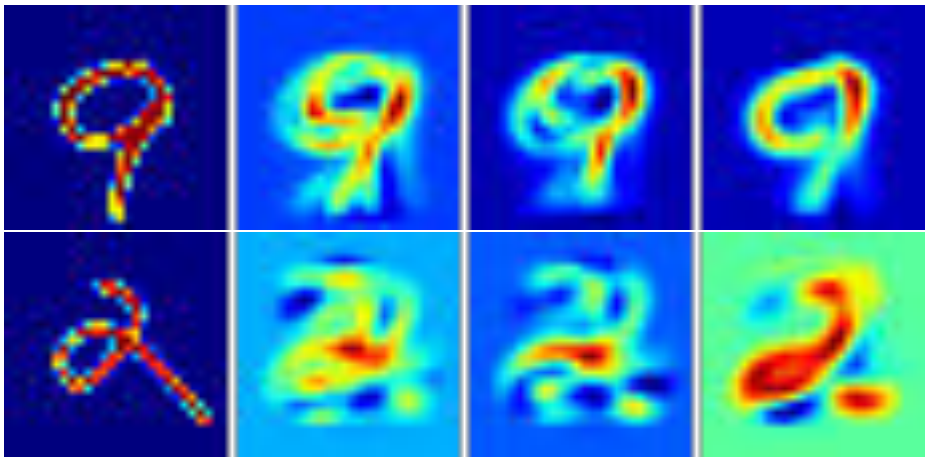


Figure 5.4: Original MNIST images and PCA attack results over them, in 30, 20 and 10 secret keys scenario. The MNIST image in the first row is the one from the test set with the lowest MSE (2154) in a 10 secret keys scenario, and the MNIST image in the second row is the one from the test set with the highest MSE (15590) in a 10 secrets key scenario.

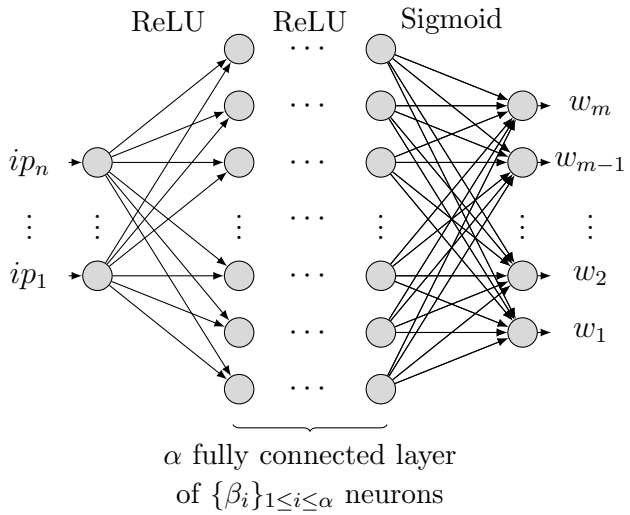


Figure 5.5: Description of the ANN model used for the ANN-attack.

last layer activation function is the sigmoid function, the ReLU function is the activation function for the other layers (details are given in Section 3.3). Figure 5.5 gives a visual description of this model. Note that we use the same notations as in the previous section.

We use the mean squared error function as the neural network training loss function.

Again, we decompose the attack on a per-class basis, so there are k ANN to train for each class c. The ANN_c denotes the class c ANN. Let $\{\vec{t}_{c,i}\}_i$ be the set of c-class vectors from the training set. The ANN_c training algorithm is run with $\{A * \vec{t}_{c,i}\}_i$ as ANN inputs and $\{\vec{t}_{c,i}\}_i$ as ANN outputs.

The attacker is now able to generate a vector relatively close to the original one, which should be kept secret. When he gets an encryption of a vector $\vec{v}$, he gets the $\vec{ip}$ vector with the IPFE decryption algorithm, then he determines its c-class and simply use its ANN_c with $\vec{ip}$ as input.

In our experimentation, the parameters used for this attack are: $\alpha = 4$, $\beta_1 = 128$, $\beta_2 = 256$, $\beta_3 = 512$ and $\beta_4 = 784$.

Examples of the attack results are given in Figure 5.6. It shows the results of our attacks on some images from the MNIST [LCB] test set. Again, rows are composed from left to right, by the original MNIST image, the result of the attack in a 30 secret keys scenario, the result of the attack in a 20 secret keys scenario and the result of the attack in a 10 secret keys scenario. We select again the two images from the MNIST test set that has for the first row the lowest MSE (72) in a 10 secret keys scenario and for the second the image with the highest MSE (10093) in a 10 secret keys scenario.

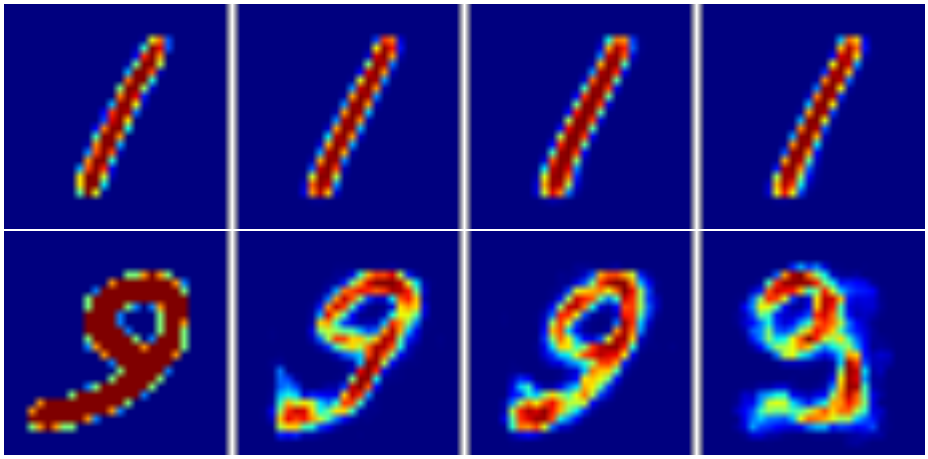


Figure 5.6: Original images and fully connected ANN attack results over them, in 30, 20 and 10 secret keys scenario. The MNIST image in the first row is the one from the test set with the lowest MSE (72) in a 10 secret keys scenario, and the MNIST image in the second row is the one from the test set with the highest MSE (10093) in a 10 secret keys scenario.

Visually, ANN attacks are more efficient than the PCA attack (Figures 5.3 and 5.4).

In a 30 secret keys scenario, it is clear that the leakage is enough to recover an image looking exactly like the original one, even if the shape of the digit is complex.

5.2.3 Convolutional network

This attack is equivalent to the previous one, except that it is using a different structure for the neural network. Since we are using the MNIST [LCB] database to experiment our attacks, we consider using Convolutional Neural Networks (CNN) which tend to have better performances over images.

The convolutional neural network we use, shown in Figure 5.7, has ten hidden layers, and the input is a vector of length n, the inner products $ip_1, ..., ip_n$. The layers used are either fully connected, convolutional, max pooling or upsampling layers. This CNN is inspired by the autoencoder CNN model suggested in Keras [C+15] for image denoising.

- *The first hidden layer* is a fully connected layer that converts the small vector into a two dimensional array of size 28×28.
- *The second layer* is a convolutional layer with 32 feature maps. Each unit in each feature map is connected to a 3×3 neighborhood in the

input. The size of the feature maps is the same as the input size because the area outside of the input, that is needed to compute the convolution, is padded with zeros. The activation function is the ReLU function.

- *The third layer* is a max pooling layer. Each unit in each feature map is connected to a 2×2 neighborhood in the previous layer and its value is the maximum value of this neighborhood. The feature maps have now a size of 14×14.

- *The fourth layer* is another convolutional layer with the same characteristics as the second layer, except that the feature maps size is 14×14.

- *The fifth layer* is another max pooling layer with the same characteristics as the third layer. The feature maps have now with a size of 7×7.

- *The sixth layer* is another convolutional layer with the same characteristics as the second layer, except that the feature maps size is 7×7.

- *The seventh layer* is an upsampling layer. It duplicates the rows and columns of the features maps. The feature maps have now a size of 14×14.

- *The eighth layer* is another convolutional layer with exactly the same characteristics as the fourth layer.

- *The ninth layer* is another upsampling layer which is exactly the same as the seventh layer. The feature maps have now a size of 28×28.

- *The tenth layer* is the last convolutional layer with only one feature map of size 28×28. Each unit in the feature map is connected to a 3×3 neighborhood in the previous layer. The activation function is the sigmoid function.

We use the mean squared error function as the neural network training loss function.

The attack is also split into k parts, one for every c-class. Therefore, there are k CNN to train. The rest of the attack is exactly the same as the previous one.

Examples of the attack results are given in Figure 5.8. Since the experiment is performed over images, it is not surprising that the CNN attack is visually better than the fully connected ANN attack.

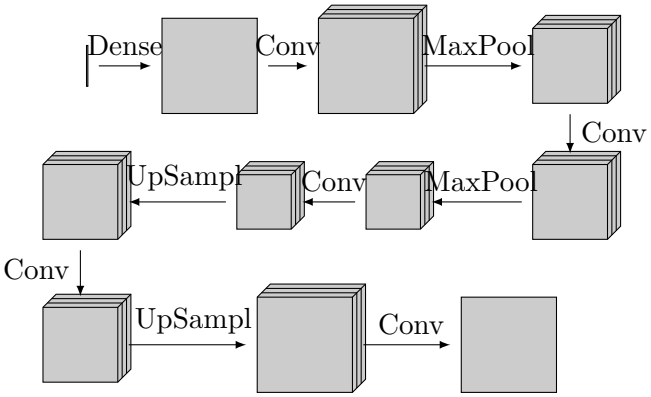


Figure 5.7: Description of the CNN model used for the CNN-attack.

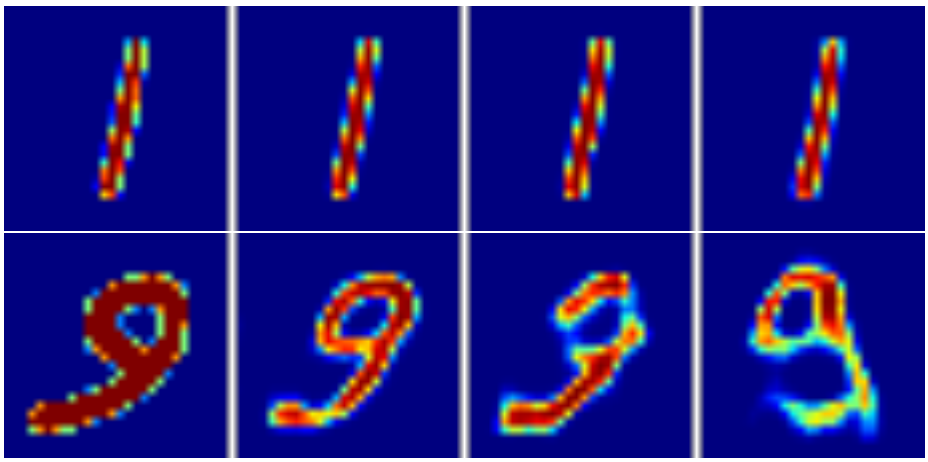


Figure 5.8: Original images and CNN attack results over them, in 30, 20 and 10 secret keys scenario. The MNIST image in the first row is the one from the test set with the lowest MSE (68) in a 10 secret keys scenario, and the MNIST image in the second row is the one from the test set with the highest MSE (12683) in a 10 secret keys scenario.

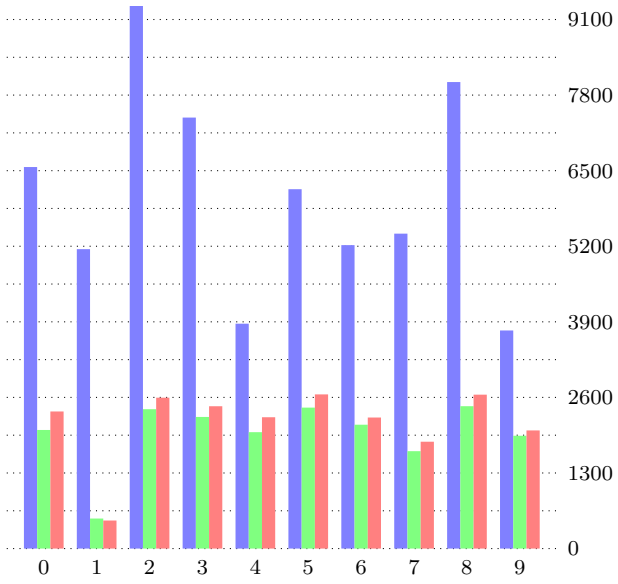


Figure 5.9: Comparison between PCA (left column), fully connected ANN (middle column) and CNN (right column) attacks in the 10 inner-product context. the x-axis shows the average MSE for all digits in abscissa.

5.3 Results analysis

The experiments were performed on a regular laptop computer with an Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz and 16GB of RAM. The Keras framework [C+15] was used to implement ANN and CNN attacks.

The *Mean Squared Error (MSE)* (details are given in Section 3.3.2) is used to compare the resemblance between original and recovered images.

5.3.1 Digit comparison

Figure 5.9 shows for each digit and for each scenario (30, 20 and 10 key) the average MSE measure between the result of each of the three attacks and the original test set. We observe that the results of fully connected ANN attack and CNN attack are almost the same in terms of MSE, even if fully connected ANN score is better by a small amount.

We can also notice from Figure 5.9, that there is an important inequality between the MSE measures obtained for different digits. Indeed, some of the digits reveal more about their shape than others. The digit “1” , for example, has the simplest shape and leaks the most, while curve shaped digits (like digit “2”, digit “5” or digit “8”) leak less and are the hardest to recover. We will now focus on the 10 secret keys scenario with the CNN attack and we

will show that it leaks already too much.

Figure 5.10 shows the MSE distribution in the CNN attack results over the test set in the 10 secret keys scenario. It also shows image examples with MSE values of 1300, 2600, 3900, 5200, 6500 and 7800 for digit "9". Attack results with MSE under 2600 give images really close to the originals, as we can see from the two image examples on the left. Such images (MSE under 2600) represent 70% of the test set.

Figure 5.11 shows for each digit, in a descending order, the percentage of images with MSE under 2600 with the CNN attack in a 10 secret keys scenario. As mentioned before, the results depend on the digits. In this figure we can see that almost every digit "1" ($> 98\%$) are "correctly" recovered with the CNN attack, while digits "2", "5" and "8" are "correctly" recovered for more than half of the examples ($< 60\%$). In a 20 secret keys scenario, the percentages are all higher than 93% and in a 30 secret keys scenario, they are higher than 97%.

Figures 5.13 to 5.22 provide for each digit and for 10 scenarios (from 1 to 10 secret keys) the result of the attack network from the neural network attack for 5 different samples and all the MSE measures. For the same sample we can notice that the MSE is not always decreasing when we increase the number of secret key. It is explained by the fact that all scenario are completely different and their training process had led to a different way to rebuilt images. The average of all the MSE measures for all the images in the test set is indeed decreasing when we increase the number of secret keys, but when we consider only one particular image, it becomes more or less efficient according to the training process performed before.

5.3.2 Privacy-preserving classification and MNIST dataset use case

Ten secret keys are leaking enough to recover every shapes of each digit "1", and more than 50% of the other digit shapes. In this case, the privacy-preserving feature seems to be compromised.

Our study is not only valid with this particular cryptographic scheme and this particular classifier. As long as an inner-product functional encryption scheme is deployed, attacks like those we presented can be used.

The more limited the plaintexts are, the more efficient the attacks are. An attacker can easily cut the plaintext space into classes, and then process as explained in the previous section to attack any ciphertext he receives. So it seems really essential to make sure that we have a precise idea of the leakage when we use such schemes.

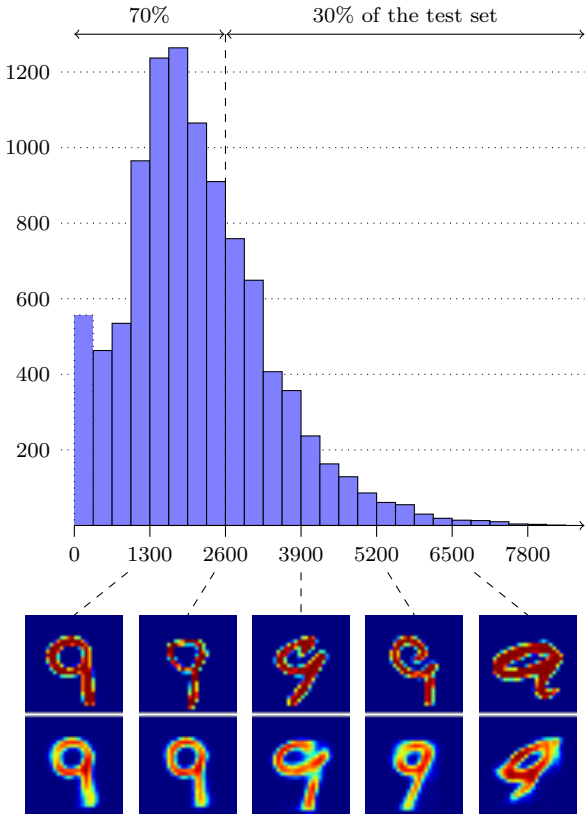


Figure 5.10: Histogram of the MSE result of the CNN attack. The x-axis represents MSE mesure, and the y-axis shows the number of image from the test set (total of $10,000$ images). We omit the results for which the MSE is larger than 8400, because it represents less than 0.1% of the test set.

1	7	9	4	6
98.50%	80.93%	75.61%	69.95%	68.99%

0	3	2	5	8
66.93%	64.55%	57.17%	56.05%	55.13%

Figure 5.11: For each digit, percentage of images from the test with an MSE under 2600 with the CNN attack in a 10 secret keys scenario (ordered in descending order).

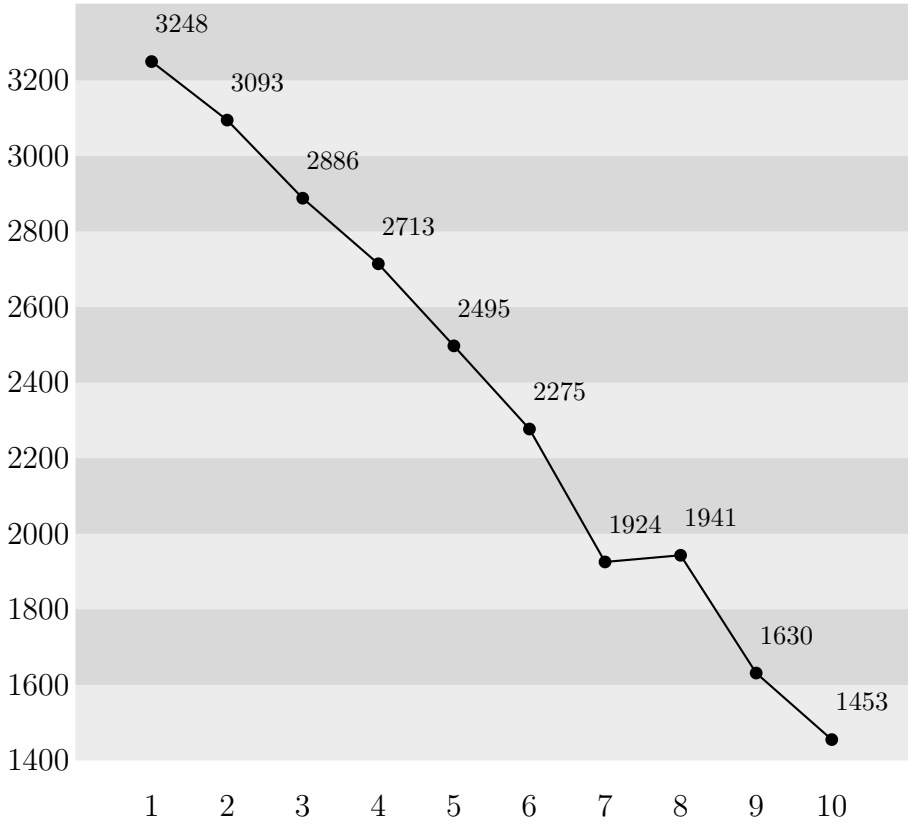


Figure 5.12: For each scenarios between 1 secret key to 10, the average MSE for the MNIST test set.

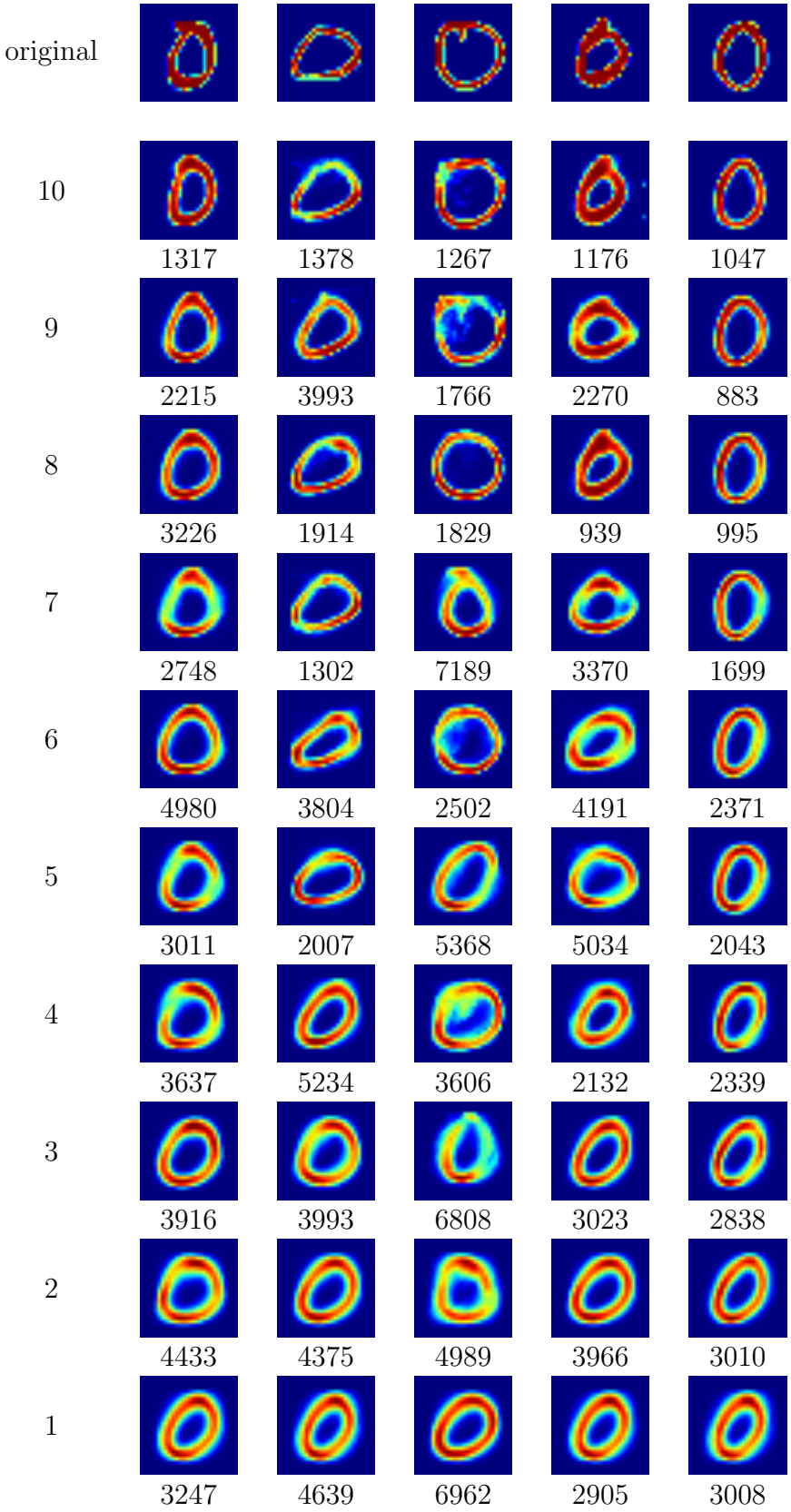


Figure 5.13: Five samples from the MNIST test set of 0 digit images and the result of the ANN attack on it for 1 secret key scenario to 10 secret keys scenario

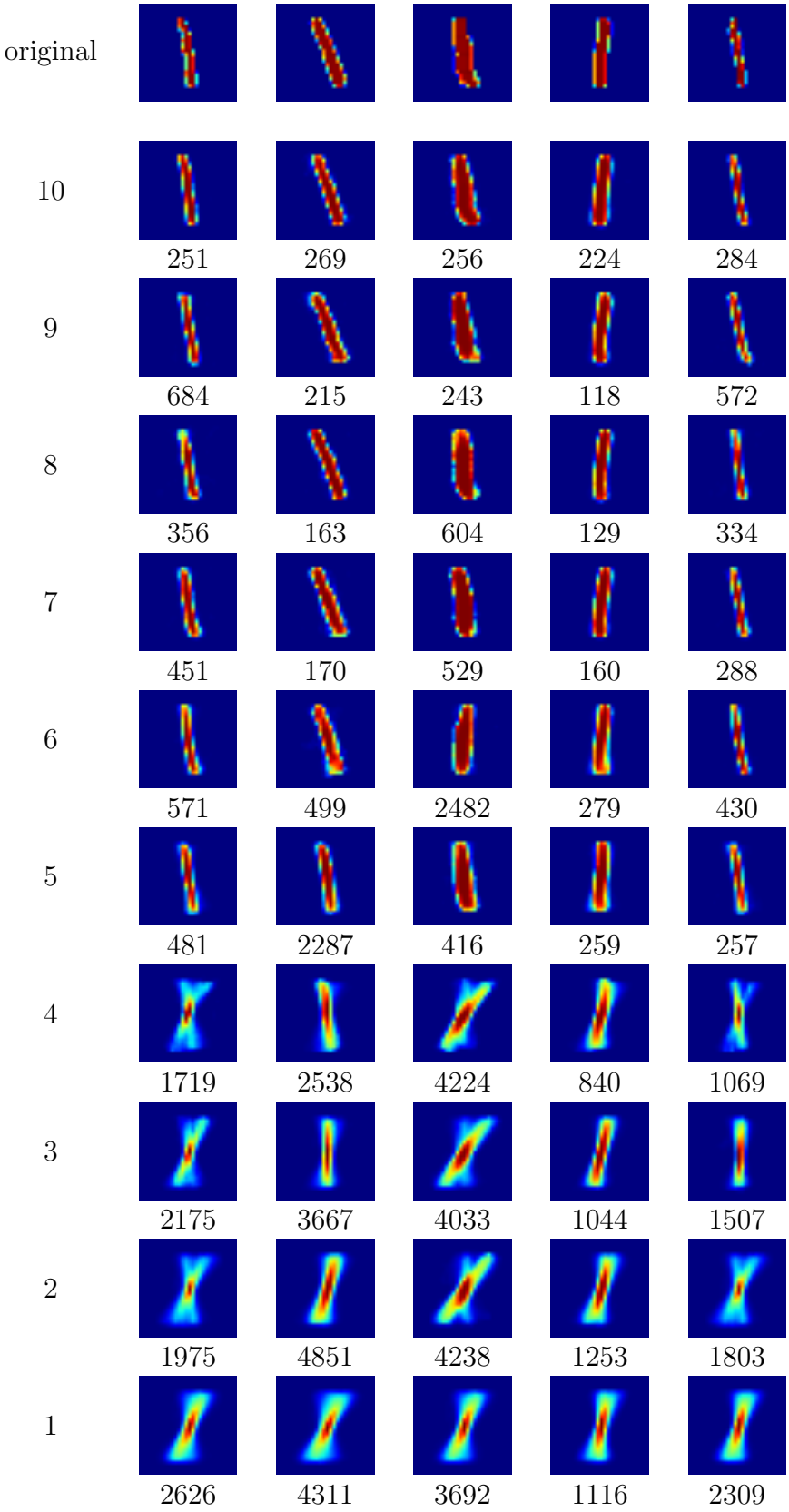


Figure 5.14: Five samples from the MNIST test set of 1 digit images and the result of the ANN attack on it for 1 secret key scenario to 10 secret keys scenario

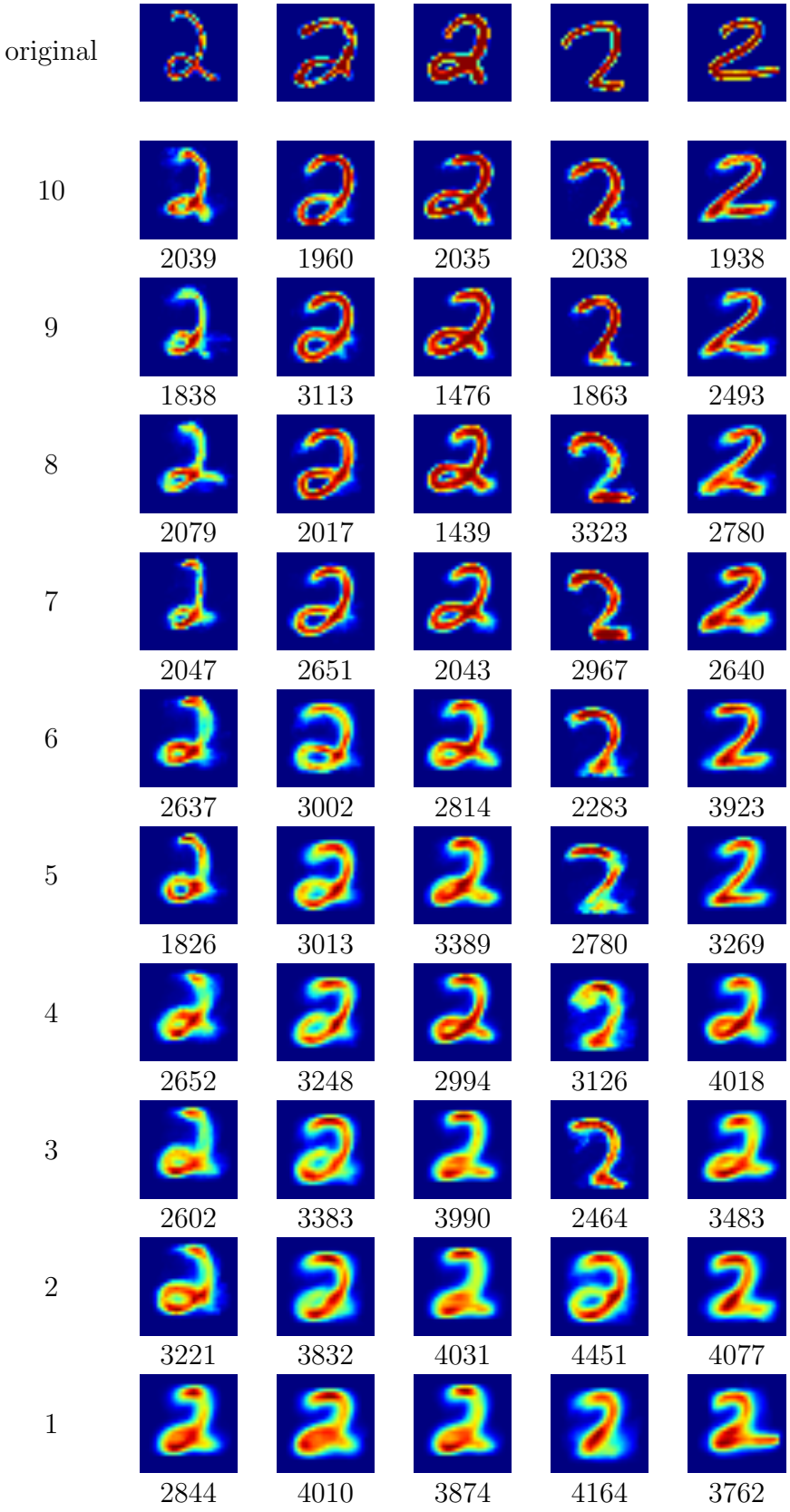


Figure 5.15: Five samples from the MNIST test set of 2 digit images and the result of the ANN attack on it for 1 secret key scenario to 10 secret keys scenario

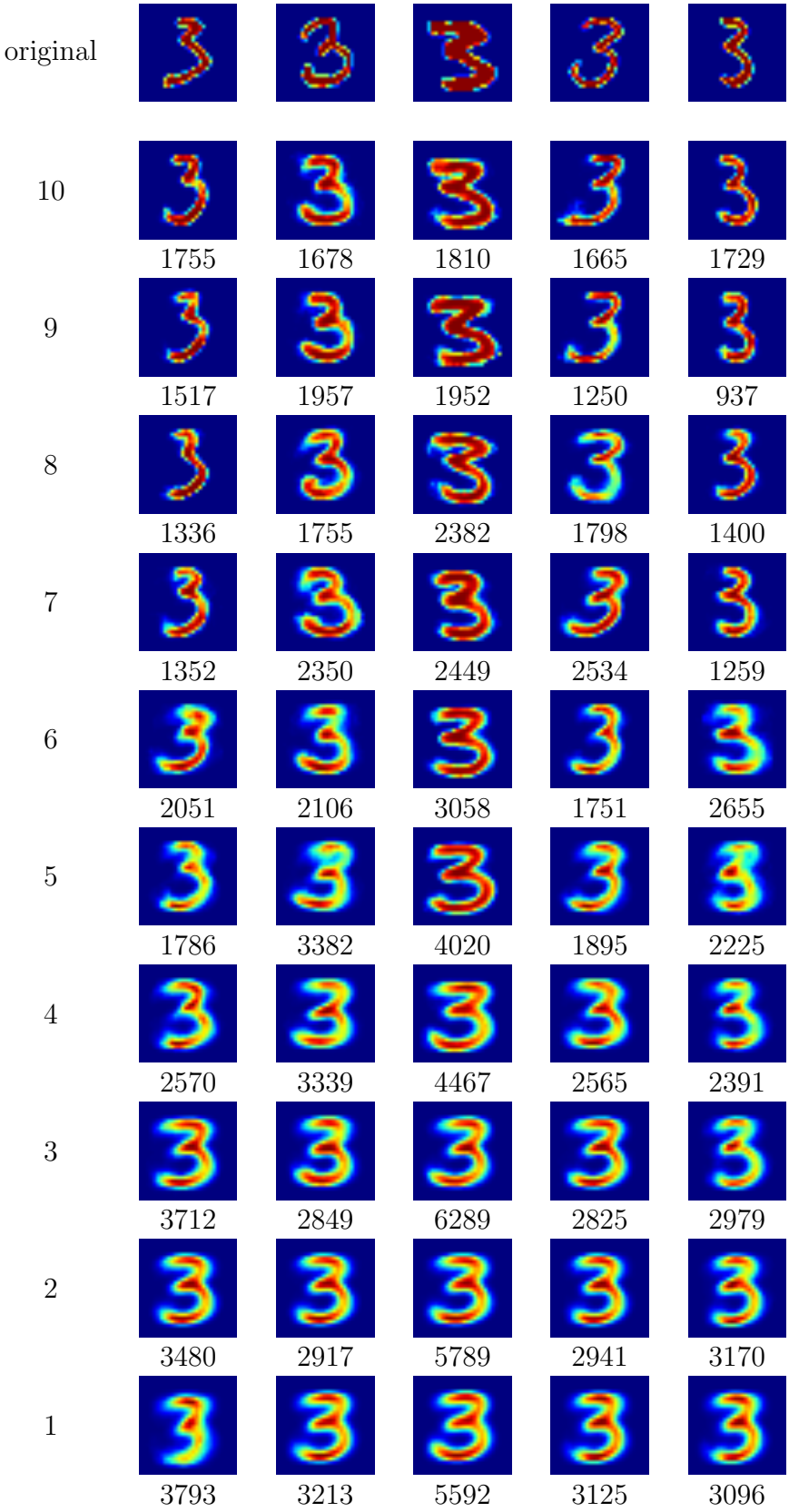


Figure 5.16: Five samples from the MNIST test set of 3 digit images and the result of the ANN attack on it for 1 secret key scenario to 10 secret keys scenario

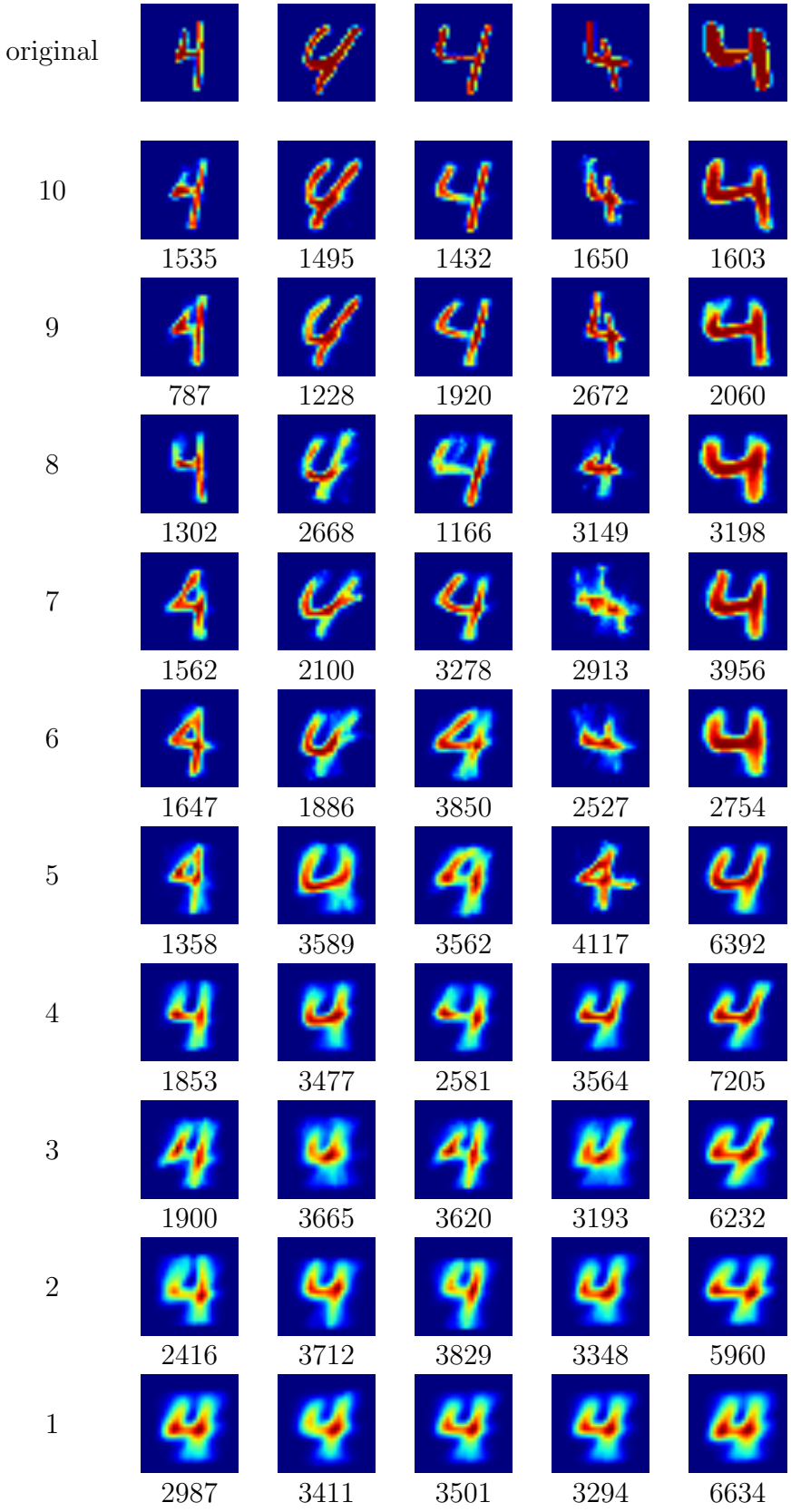


Figure 5.17: Five samples from the MNIST test set of 4 digit images and the result of the ANN attack on it for 1 secret key scenario to 10 secret keys scenario

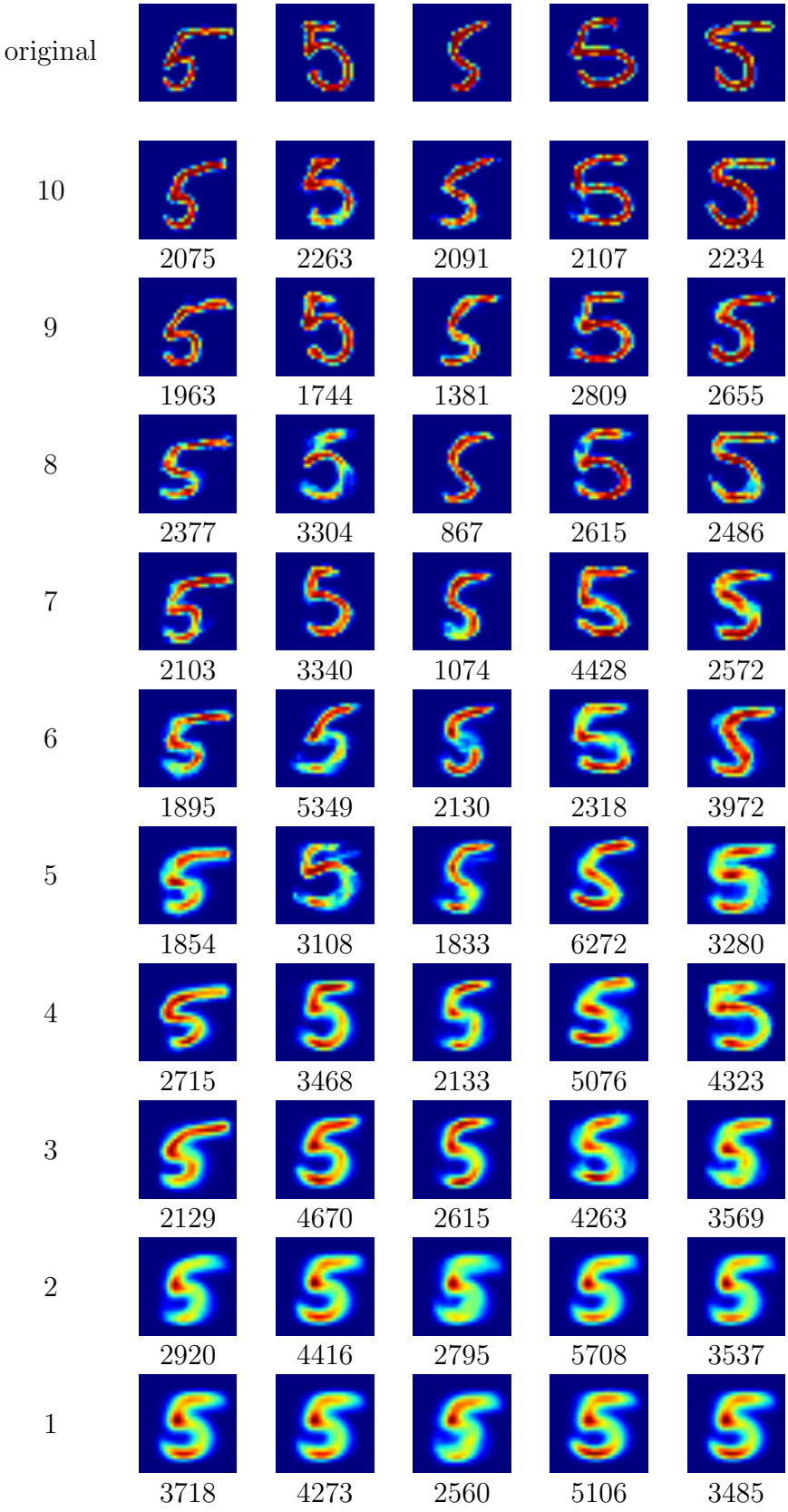


Figure 5.18: Five samples from the MNIST test set of 5 digit images and the result of the ANN attack on it for 1 secret key scenario to 10 secret keys scenario

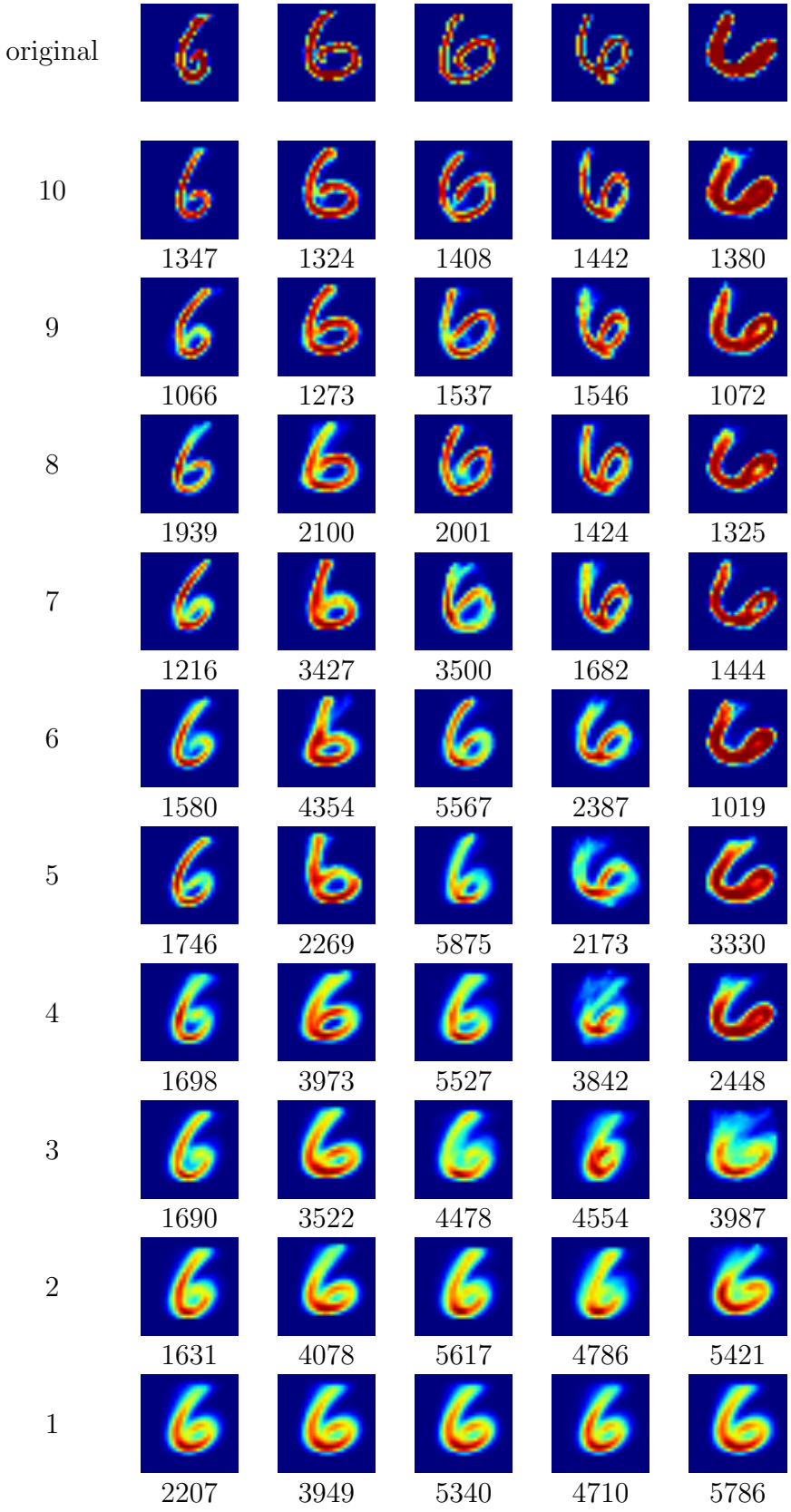


Figure 5.19: Five samples from the MNIST test set of 6 digit images and the result of the ANN attack on it for 1 secret key scenario to 10 secret keys scenario

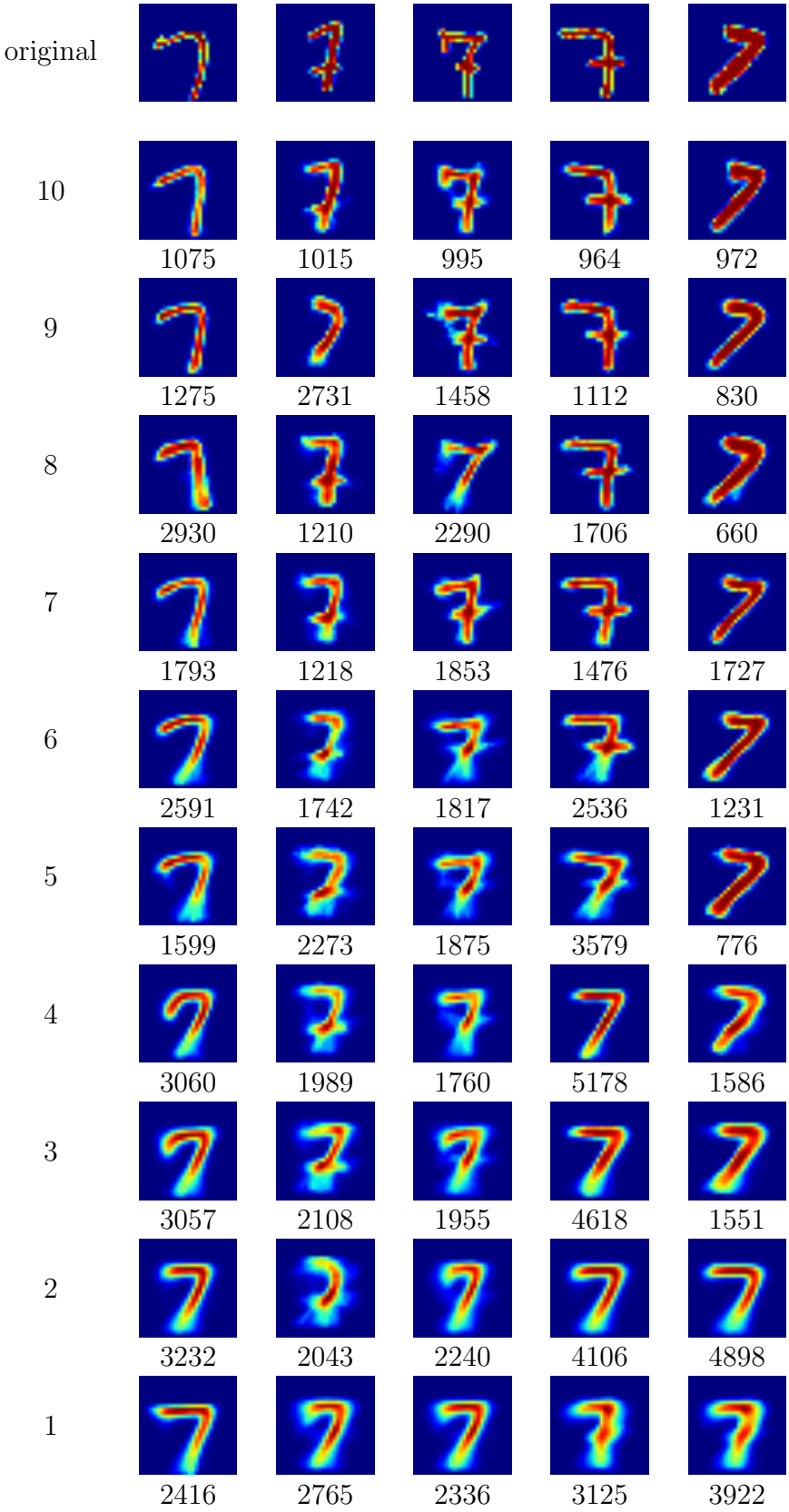


Figure 5.20: Five samples from the MNIST test set of 7 digit images and the result of the ANN attack on it for 1 secret key scenario to 10 secret keys scenario

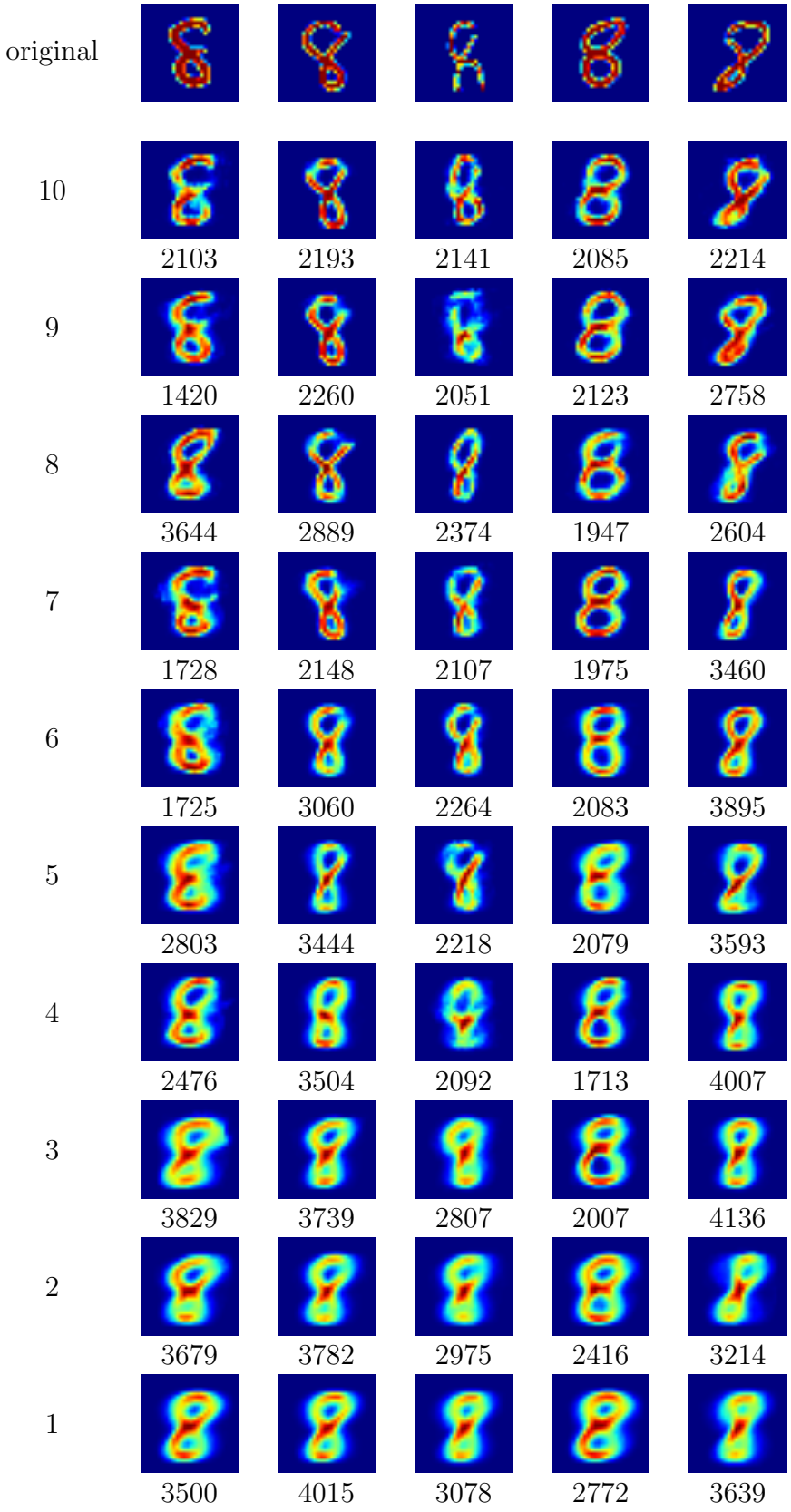


Figure 5.21: Five samples from the MNIST test set of 8 digit images and the result of the ANN attack on it for 1 secret key scenario to 10 secret keys scenario

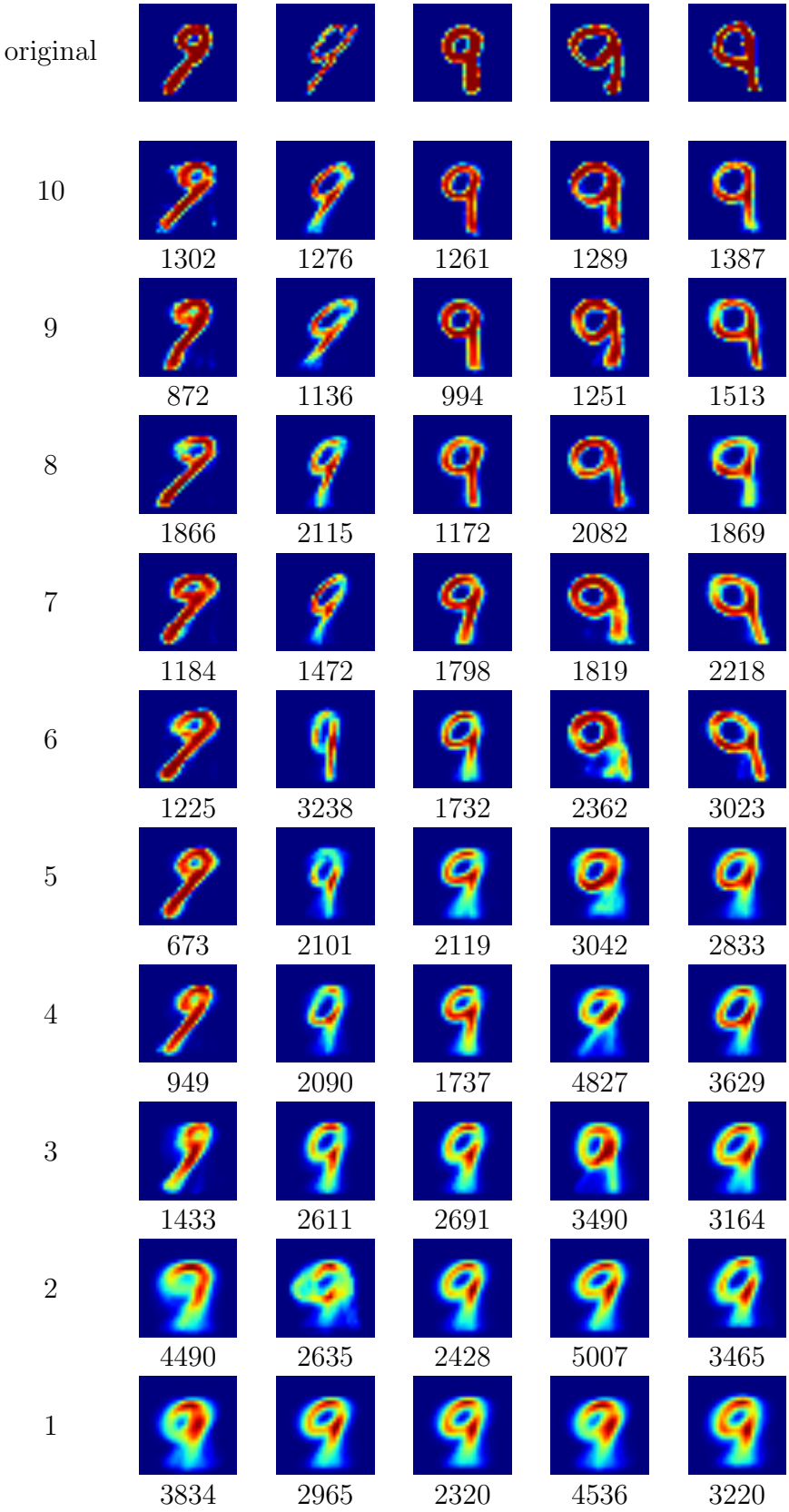


Figure 5.22: Five samples from the MNIST test set of 9 digit images and the result of the ANN attack on it for 1 secret key scenario to 10 secret keys scenario

5.4 Generalization to others functional encryption schemes

The neural network based method to analyse the leakage of an IPFE scheme can be used for any other FE scheme. Indeed, even if the functional encryption scheme is used for something else than classification, those attacks may still be fitting. When plaintexts are random, it has the worst efficiency, but as soon as plaintexts are sharing some properties, those attacks, can exploit those similarities.

As we found out with our quadratic polynomial functional encryption experimentation (in 4.4), increasing the degree of the polynomial function evaluation makes classification easier. Our intuition is that it leaks more which makes an attack also easier. We need schemes which are also design to compute maximum/minimum functions or argmax/argmin functions for example. In fact, it would definitely make the leakage harder to exploit.

5.5 Discussion

We just proposed attacks designed to exploit inner-product functional encryption's leakage. The attack based on neural networks is the most efficient. It can be shifted to whatever functional encryption scheme.

In our particular use case, which is classification of digit images, thus appear to be quite easy to get a good approximation of most of the plaintexts. The more secret keys are provided, the better is the approximation. For instance with ten secret keys from an IPFE, one is able to built a good approximation for more than half of the digit images!

This is what we want to stress in this thesis: a cryptographically-secure functional encryption scheme may not achieve the desire security goeal depending on the use case. These attacks represent a way to estimate this risk. We go further in the next chapter by trying to decrease the attack efficiency while keeping a high classification performance.

Chapter 6

Tradeoff between classification performance and attack efficiency

Chapter 4 shows how to combine functional encryption and machine learning in order to perform classification over encrypted data. Chapter 5 proposes ways to attack a practical and secure functional encryption such as the use case described in the previous chapter. This chapter aims to manage the amount of information released by the functional encryption scheme. In our privacy-preserved classification use case, we must classify efficiently (chapter 4) but also resist to attacks (chapter 5).

The leakage depends on the secret keys. In an inner-product functional encryption case, it depends on vectors associated with the generated secret keys. The challenge is to find vectors that provide the best classification performance and the worst attack efficiency.

In this chapter we investigate the trade-off between classification performance and attack efficiency for those vectors. At first we recall how we got our vectors that give the best classification performance, then we take an interest in random vector sets. We later propose a way to generate "better" vectors using a generative adversarial neural network (GAN). We finally discuss our results at the end of this chapter.

6.1 Vectors with the best classification performance

We generated vector sets in chapter 4 that offer the best performance of classification. They are chosen using a neural network that has for objective to

classify digits. This network is composed by the inner-products computation and then a fully-connected network as in Figure 4.8. For those vector sets, figures from 6.2 to 6.11 show (for each digit) the performance of the classification and the efficiency of the fully connected neural network attack (note that Figure 6.1 clarify how to read those figures). We consider 10 different sizes n of the vector sets $\{w_1, ..., w_n\}$: from 1 to 10. We represented it on the figures with blue "B0" to "B10" inscriptions. As we can see the more we have vectors (secret keys) the better is the classification but also the more efficient is the attack.

6.2 Generating vectors with different approaches

To get a better idea of the spectrum where vectors are, we try to classify and to attack with random vectors. It shows what we get in the spectrum when we use vectors that are absolutely not designed for this classification task. We then use a generative adversarial neural network to explore a different area of that spectrum.

6.2.1 Random vectors

As mentioned before, it seems that vectors $w_1, ..., w_n$ may influence the system leakage. Indeed, the zero vector will leak nothing, ten collinear vectors will leak as much as just one of them, etc. So we studied sets of random vectors to determine the performance of the classification and the efficiency of the attack. We considered three cases: 1,5 and 10 vectors. For each case our result is the average of the 5 different generated instances of random vector sets. Figures from 6.2 to 6.11 show those results for all the digits (note that Figure 6.1 clarifies how to read those figures). The random sets are represented by red "R1", "R5" and "R9" inscriptions for 1, 5 and 9 vectors.

As we can see, random sets makes the classification performance worse and also the attack efficiency worse. It means that vector sets can provide very different classification performance and attack efficiency. It makes sense that random vectors are less useful than vectors design to classify those kind of images. The question is wheter or not there exist vector sets that classify well and do not lend themselves to the attack.

6.2.2 GAN obtained vectors

As we see with the random vector sets and the best vector for classification sets, just n, the number of vectors $w_1, ..., w_n$ does not determine the per-

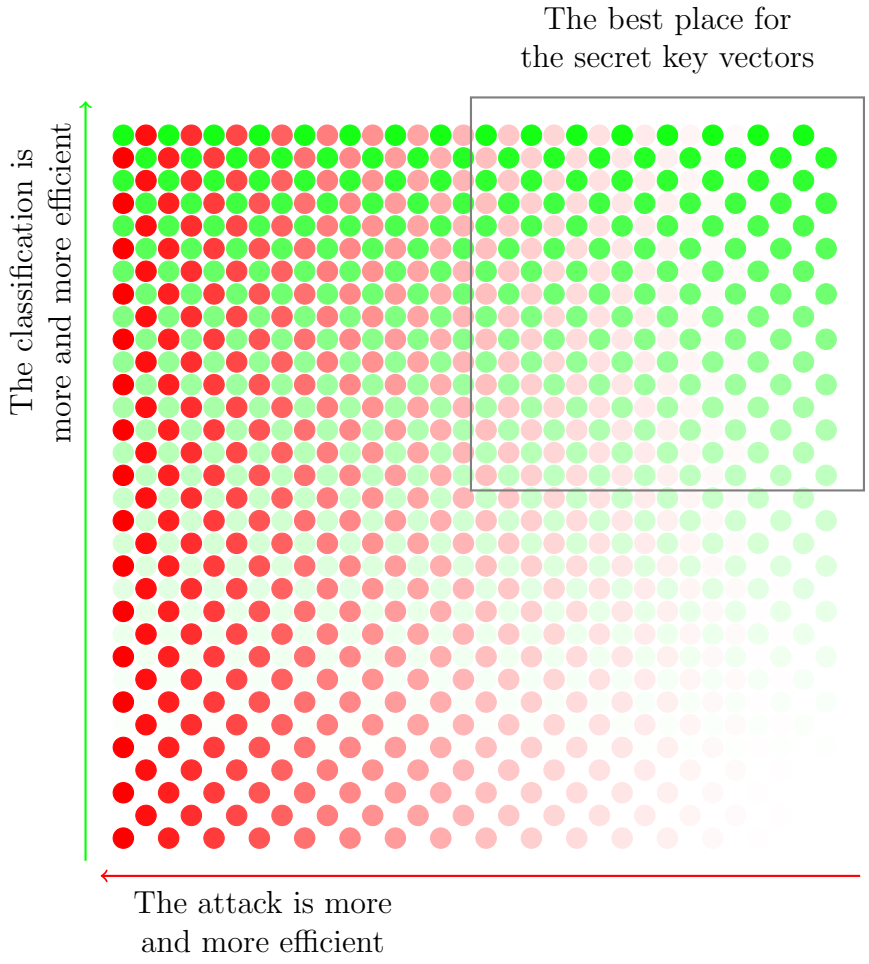


Figure 6.1: How to read the next figures. The gradation of green symbolizes the y-axis and that higher a point is the better the classification is. The gradation of red symbolizes the x-axis and that the more red it is, the better the attack works. Then we are looking for vectors in the area with just the color green (up right corner).

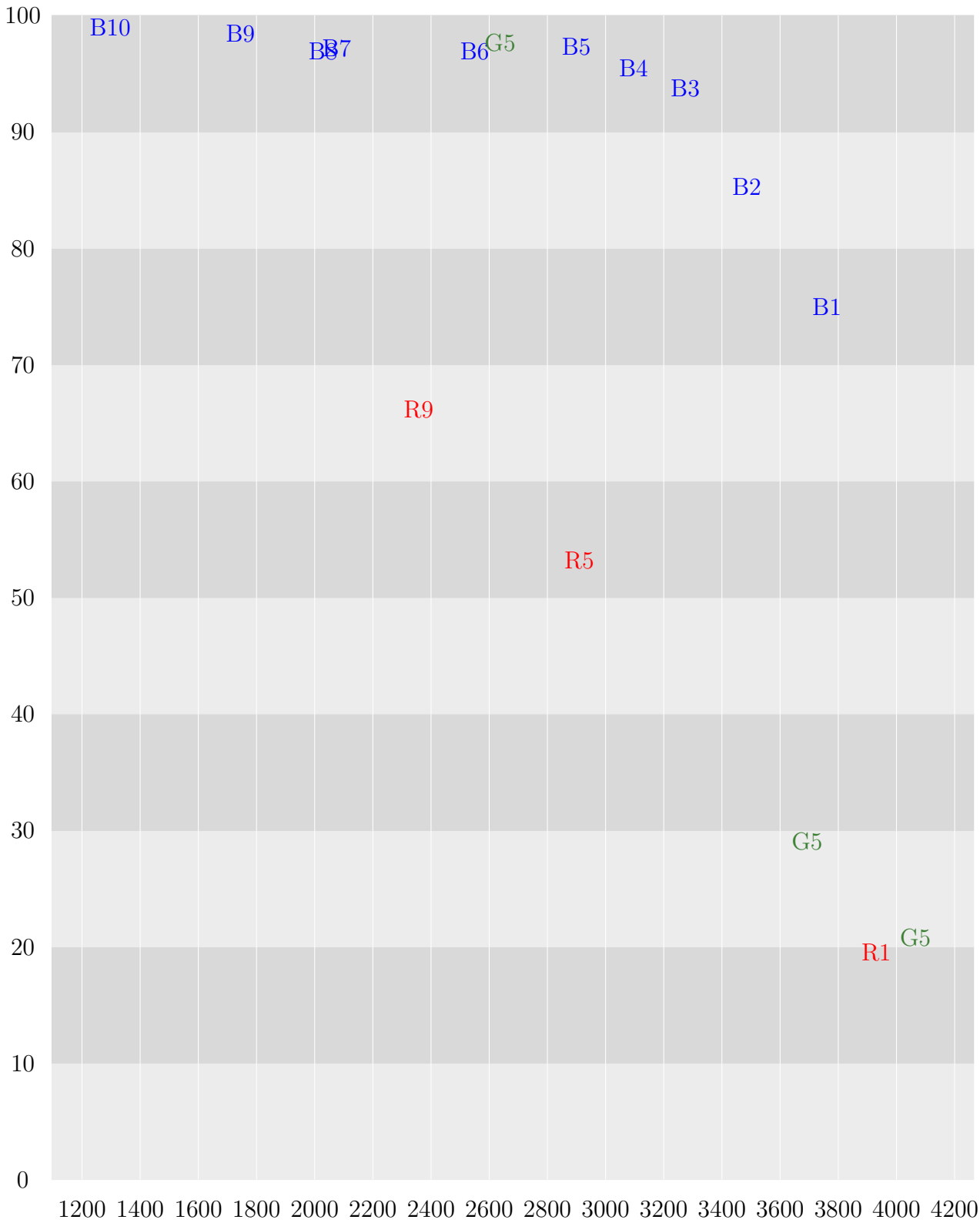


Figure 6.2: Classification performances VS mean of the MSE with the attack for the 0 digit

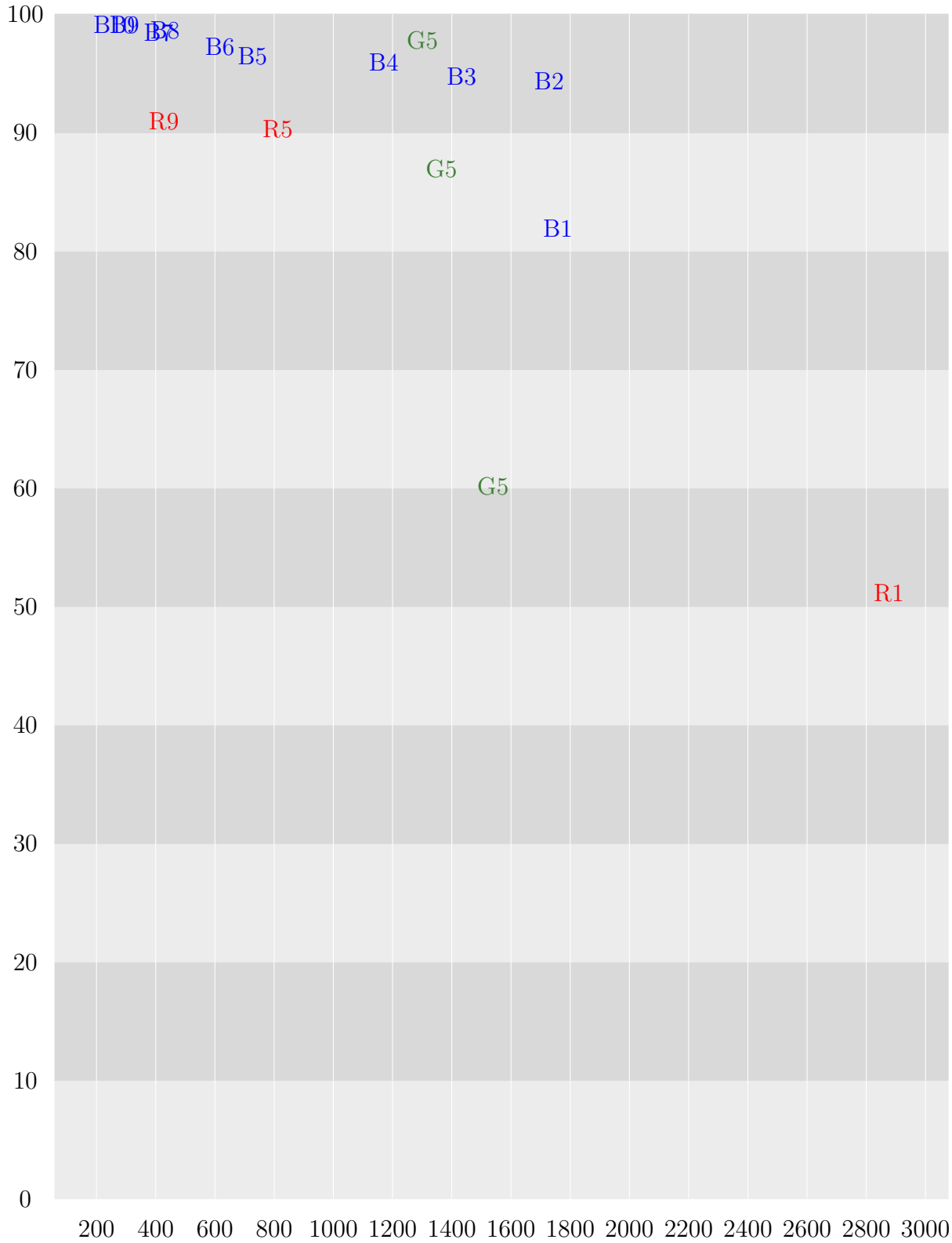


Figure 6.3: Classification performances VS mean of the MSE with the attack on the 1 digit

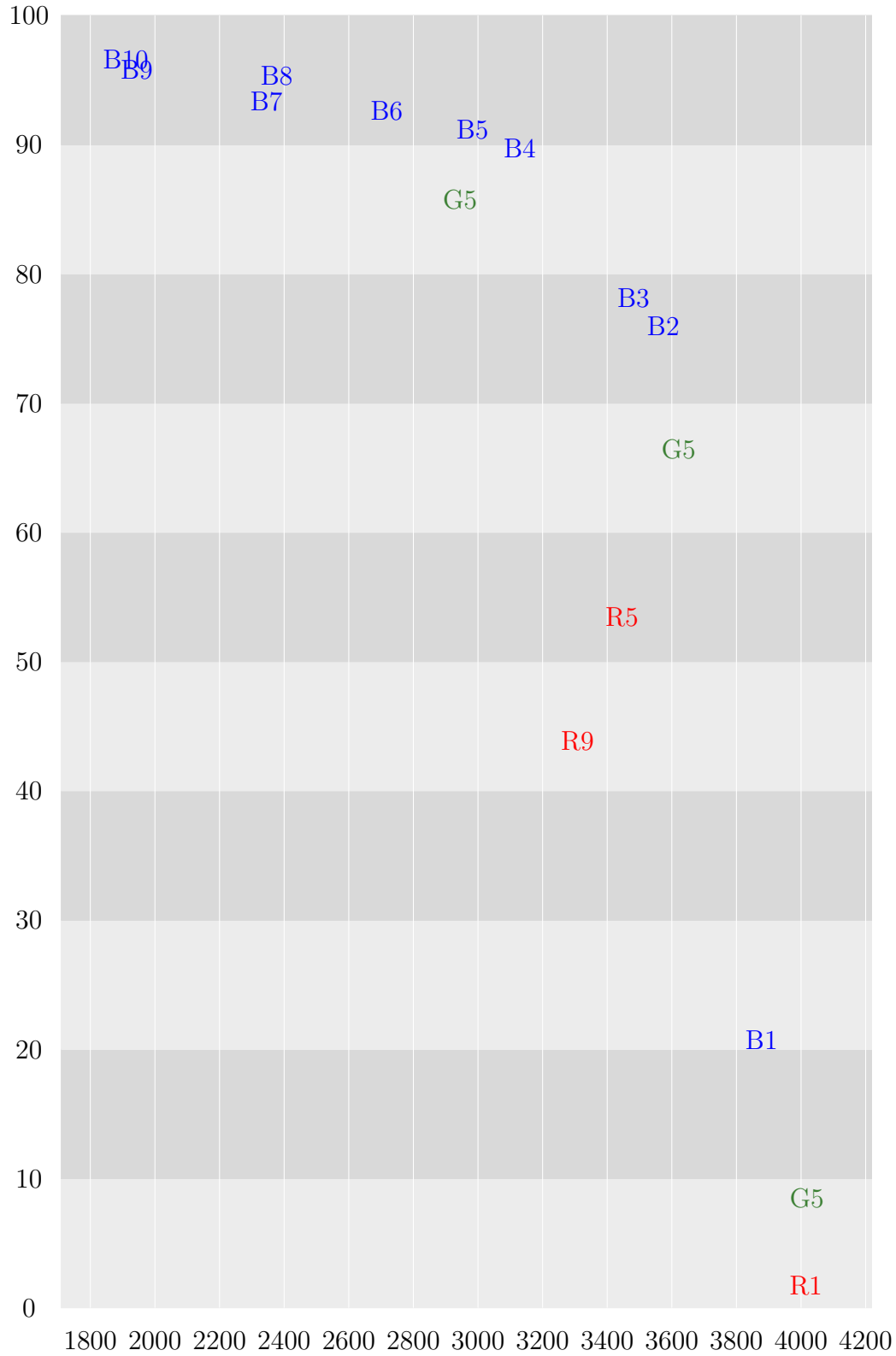


Figure 6.4: Classification performances VS mean of the MSE with the attack for the 2 digit

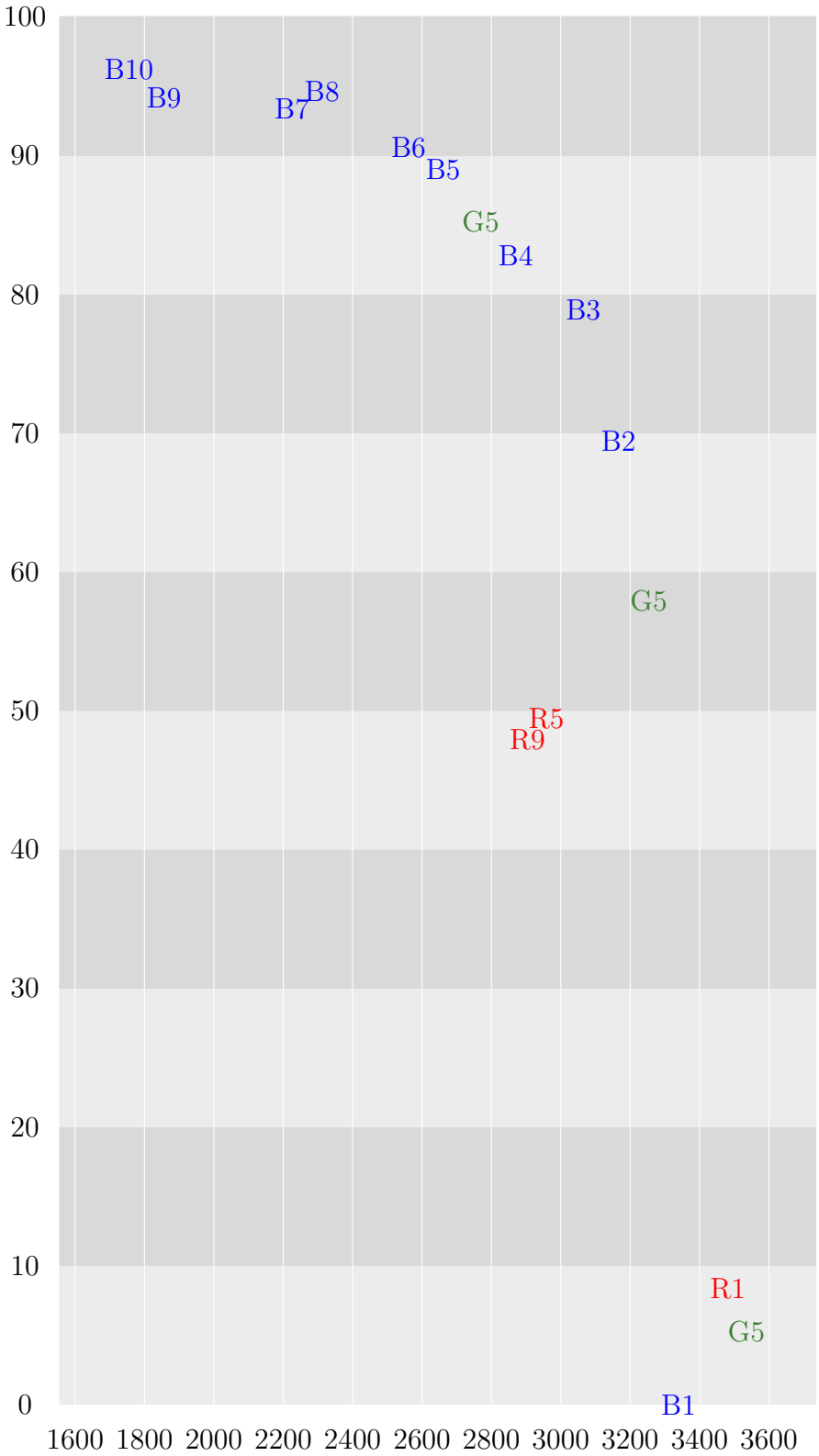


Figure 6.5: Classification performances VS mean of the MSE with the attack for the 3 digit

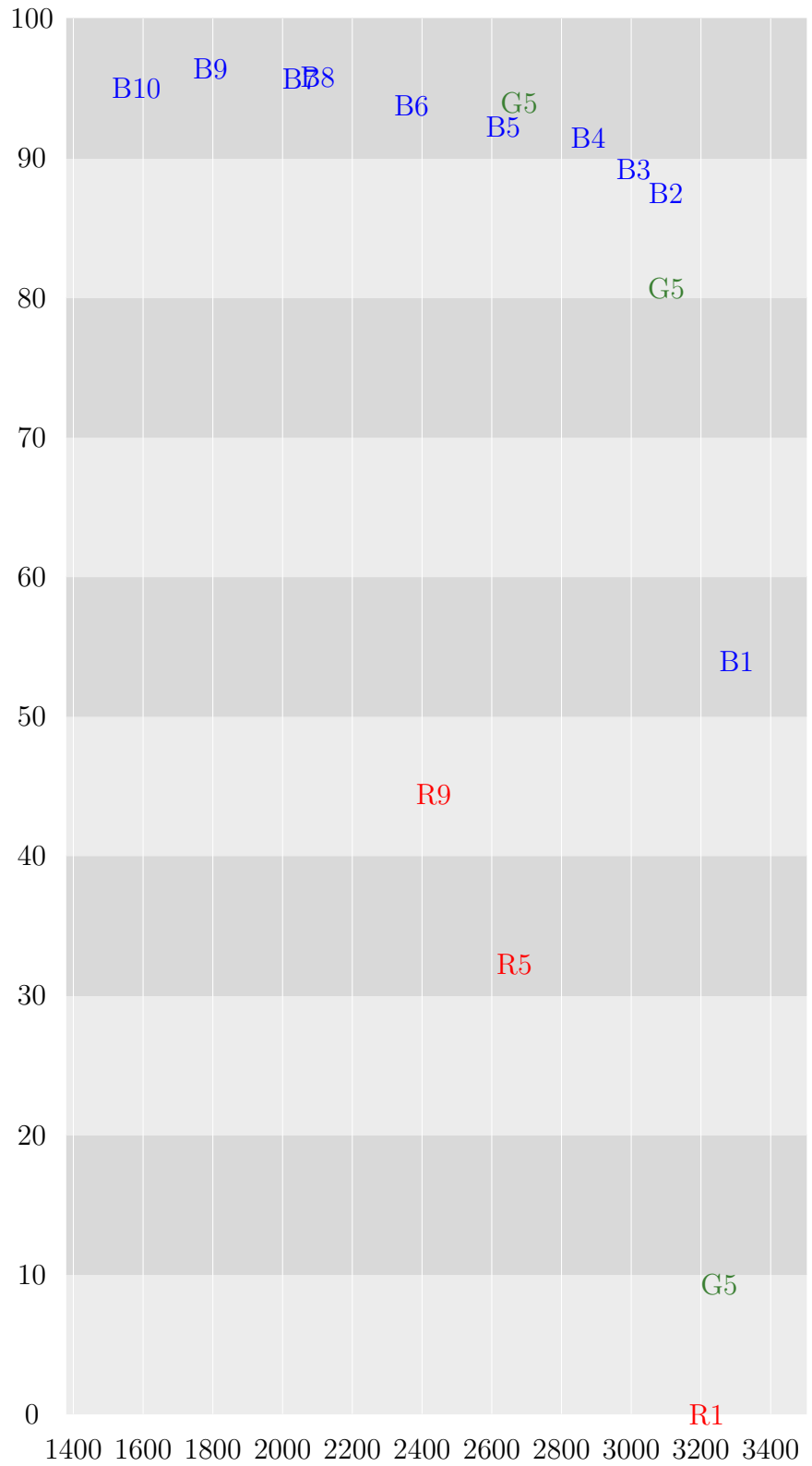


Figure 6.6: Classification performances VS mean of the MSE with the attack for the 4 digit

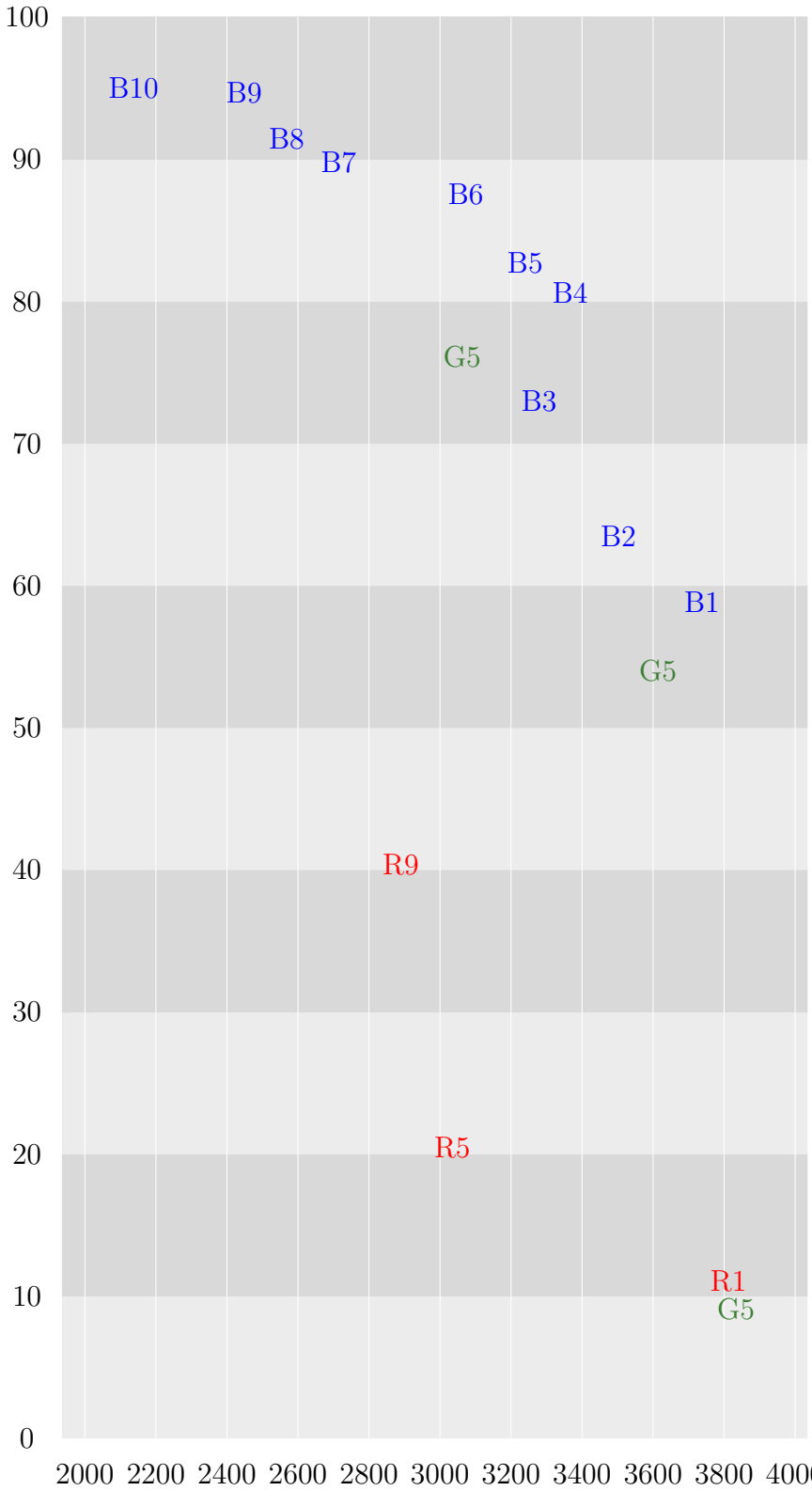


Figure 6.7: Classification performances VS mean of the MSE with the attack on the 5 digit

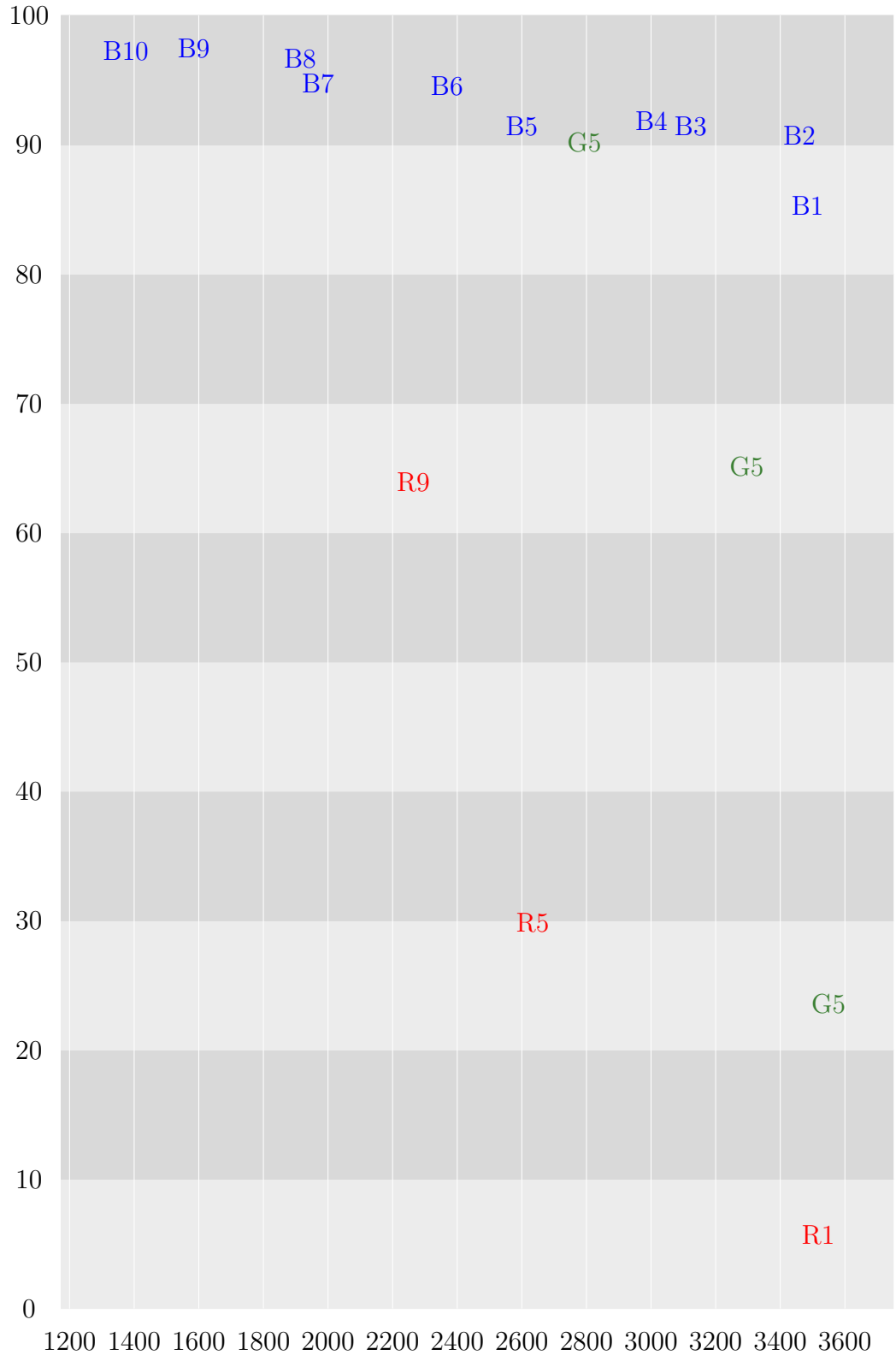


Figure 6.8: Classification performances VS mean of the MSE with the attack for the 6 digit

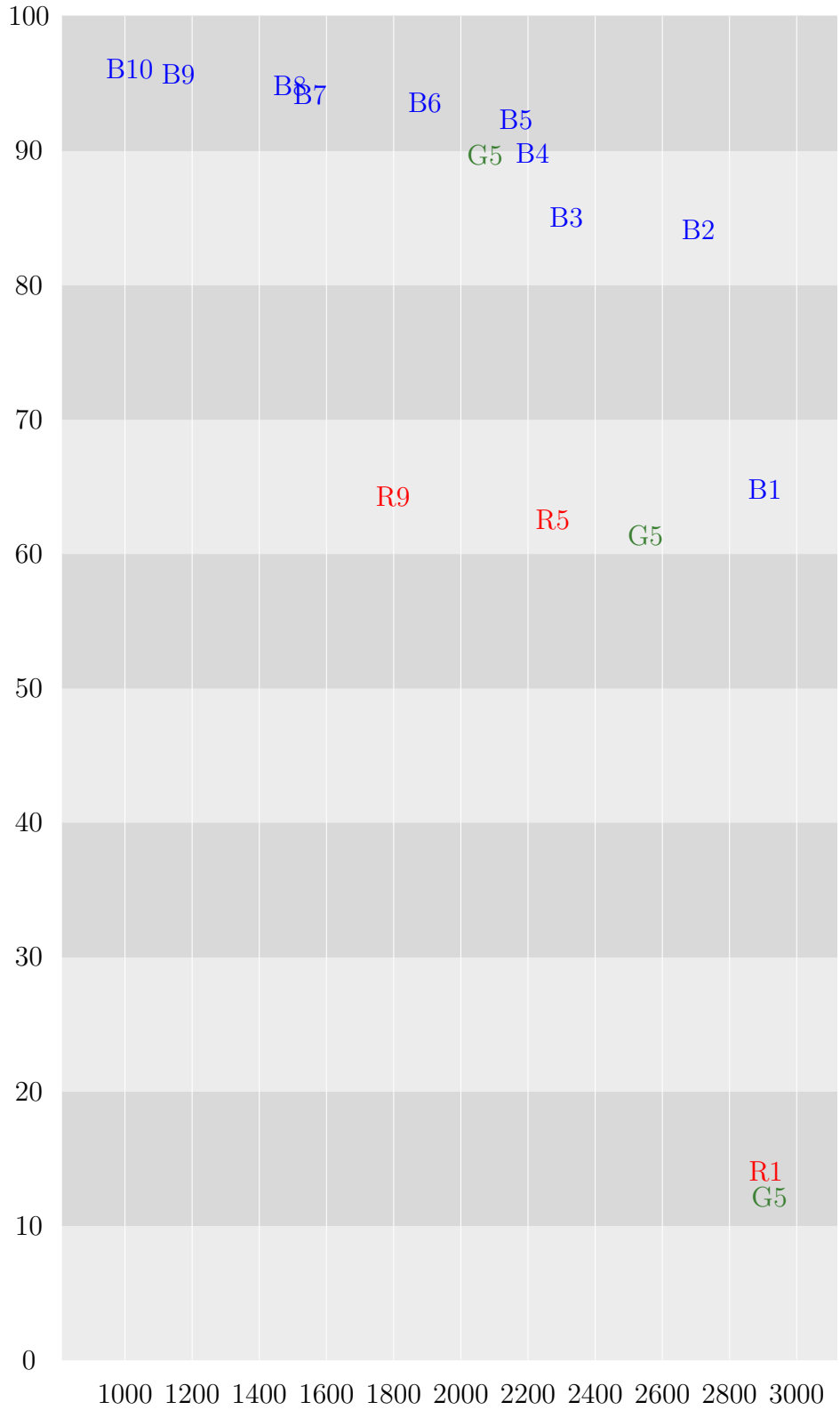


Figure 6.9: Classification performances VS mean of the MSE with the attack for the 7 digit

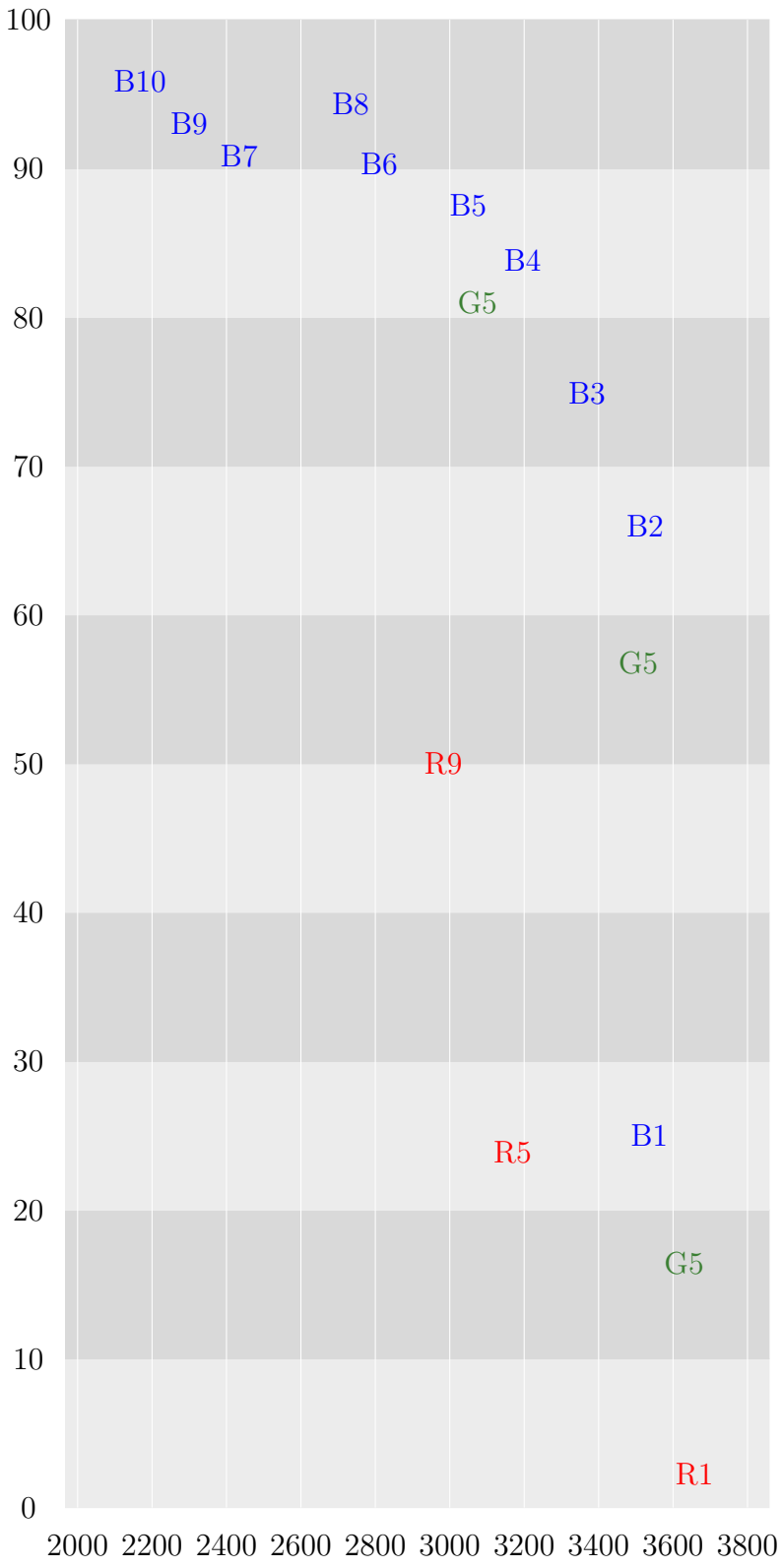


Figure 6.10: Classification performances VS mean of the MSE with the attack for the 8 digit

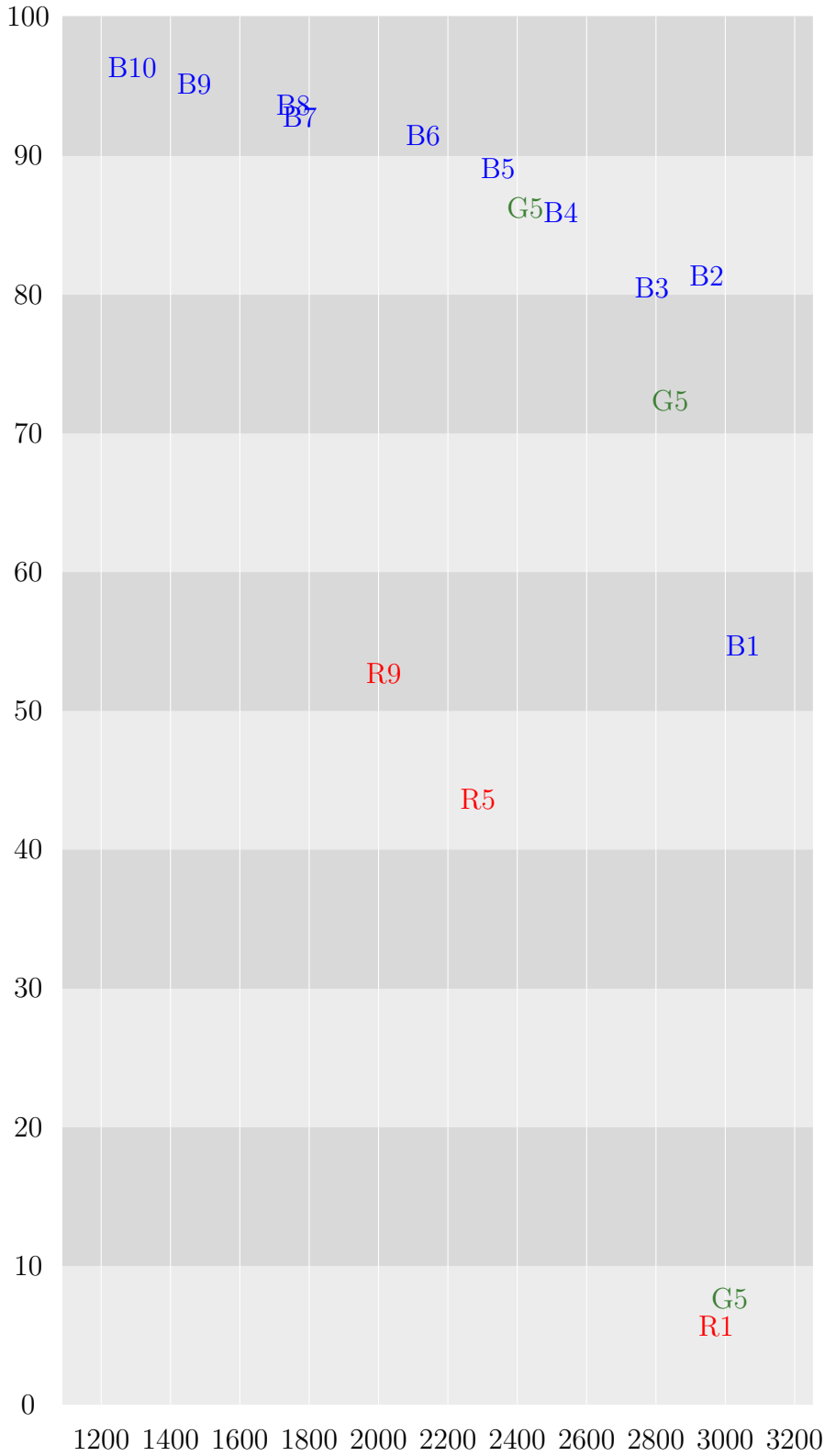


Figure 6.11: Classification performances VS mean of the MSE with the attack for the 9 digit

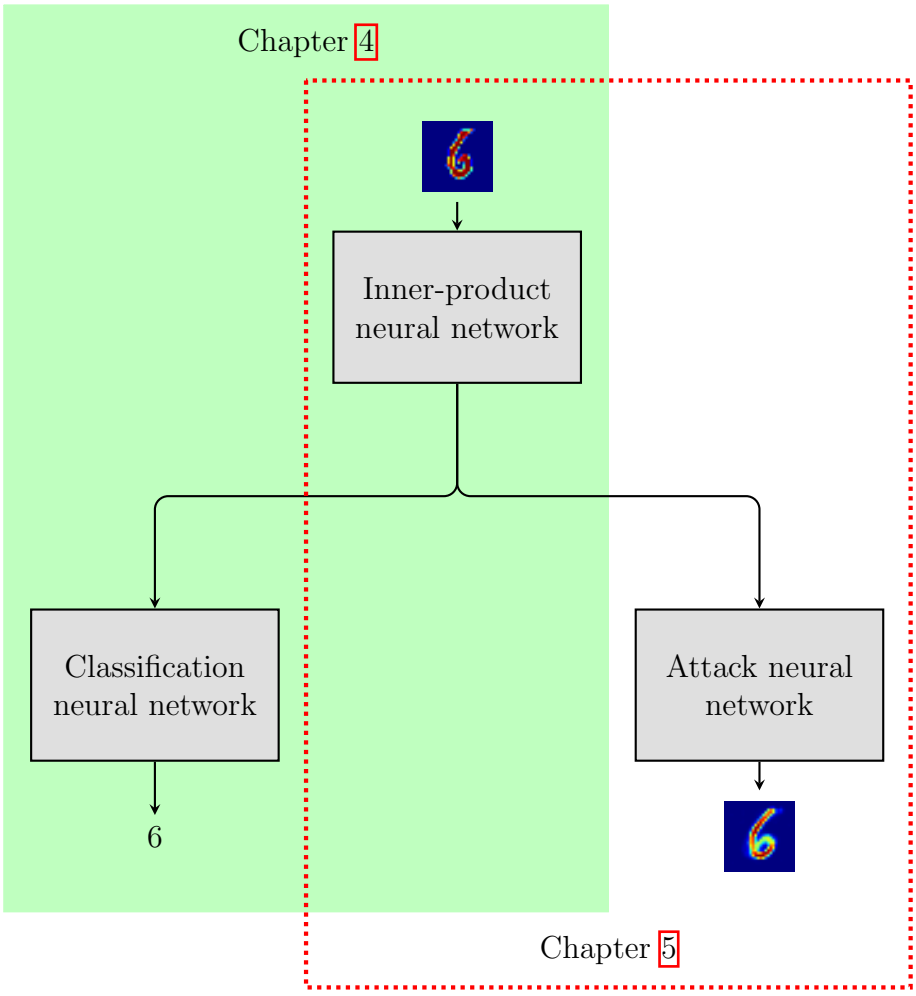


Figure 6.12: Overview of the GAN network architecture.

formance of the classification or the efficiency of the attack. We now want to visualize more precisely what happens when the number of vectors is fixed. This is why we used a Generative Adversarial Networks (GAN) to built different sets of 5 vectors with different properties for the classification and the attack. For details about GAN please refer to Section 3.3.3

Our GAN is composed by 2 players: "the classifier" and "the attacker". When it is the classifier turn to be trained, its loss function aim to reduce the distance between the actual label of the image and the predicted label. When it is the attacker turn to be trained, its loss function aim to make the attack less efficient, which means to increase the distance between the original image and its reconstruction by the attack neural network.

An overview of our GAN architecture is provided in Figure 6.13. The

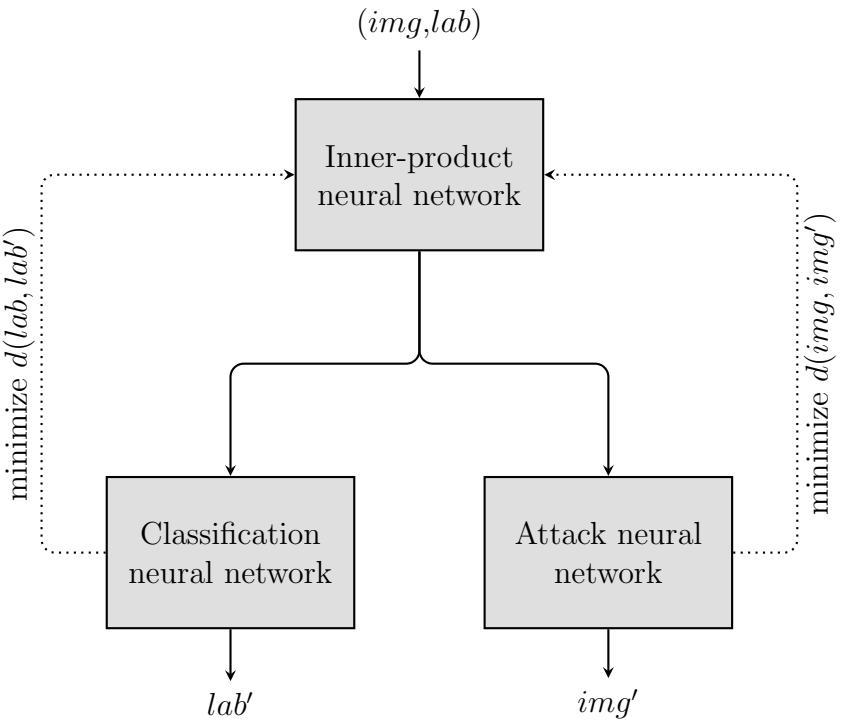


Figure 6.13: The GAN network architecture and its training objectives.

"classification neural network" refers to the network described in Figure 4.8 with the same settings that we used in Figure 4.3.4. The "attack neural network" refers to the network described in Figure 5.5 but this time we use $\alpha = 5$ layers with respectively 128, 512, 3136, 6272 and 784 neurons. This attack network is not specialized for a particular digit, it is general and aim to rebuilt all digit images.

There is an other parameter for the training process of our GAN which determine the compromise between our 2 players objectives. We experiment the GAN network for 3 different settings, the first one favors the classification performance over attack resistance, the second one favors none of the players, and finally the third one favors attack resistance over classification efficiency. Those 3 settings count only 5 secret keys.

Those results are shown in Figures 6.2 to 6.11 (note that Figure 6.1 clarifies how to read those figures). They give for each digit the performance of the classification (Y axis) and the efficiency of the attack (X axis) for our 3 instances of a GAN (green writings), but also (as mentioned before) for 3 scenarios with random vectors sets (red writings) and finally for the 10 scenarios with the best vector sets according to classification (blue writings).

6.3 Results analysis

For an easier reading of the figures from 6.2 to 6.11 we give a visual explanation in Figure 6.1. The Y axis represents performances of classification and has a green gradation. Basically the best vectors should be in the green area. The X axis represents the efficiency of the attack and has a red gradation. We do not want our vectors to lie in the red area because it means that they are weak against attacks.

Figures from 6.2 to 6.11 show our results about both the efficiency of the attack on the X axis and the performances of the classification on the Y axis.

Random vector sets are represented in red with respectively "R1", "R5" and "R9" writings for 1, 5 and 9 secret keys (i.e. vectors). Those vectors are obviously not designed for anything specific, and it is interesting to see that they are quite bad for classifying ("R9" is around 50% and is always worth that "B2") but seem more useful for rebuilding original images with neural network attacks (most of the time it has the same x value as "B6").

As we can see every digit follows roughly the same pattern. The main difference is where it stands on the X axis. The MSE can be shifted from 1400 for the 1 digit to 3200 for the 2 digit. The blue dots are always the highest. It seems that the highest of the GAN points (green) is on the curve drawn by the blue dots. This makes sense because the blue dots only focus on

classifying and this particular green point is the one that favors classification performances over preventing an attack with 5 secret keys. We can see that "B5" is very close to this green point. The GAN point tries a little to prevent an attack, which makes it less good at classifying (shifted on the bottom a little) and better at preventing an attack (shifted a little bit on the right).

We can observe that "B5" and "R5" have quite the same x value, so do "B1" and "R1". However "B9" and "R9" have not. It suggests that for a few secret keys, having random vectors and vectors designed for classification leak the same amount of information regarding our neural networks attacks. When the number of vectors increases, 9 for example, random vectors are harder to attack.

6.4 Discussion

We investigated in this chapter the dependency between vectors (associated with IPFE secret keys), classification performance and attack efficiency. We ended up with the observation that those three are strongly linked in the sense that the better the classification performance is, the easier it is to attack and vice versa, and it is all determined by the vectors.

We saw that the random vectors are the worst in terms of both classification performances and attack efficiency which makes sense. Indeed, it seems logical that in this special use case about handwritten digit images, the pixels that are given the biggest weights, should be carefully chosen. However they are way worse for classification than resistant to neural network attacks.

When we considered the vectors providing the best classification performance, we also got that it is very easy to attack with a neural network. We needed another way to design vectors that can resist to an attacker.

This analysis leads us to generate vectors with a generative adversarial network (GAN). The idea was to target both goals: generating vectors for classification but also vectors that resist to attackers. There is a parameter within the GAN that defines the weights for those two goals. We observed that when we generate 5 vectors (i.e. secret keys) with our GAN, according to the goals weights, they can have the same measures than our 5 vectors that give the best classification performances ("B5" in figures from 6.2 to 6.11) or the same measures than our one random vector.

It suggests that it is very difficult to find vectors satisfying those two goals, and that vectors good for classification implies that they are easy to attack. We also observed that the number of vectors (i.e. secret keys) is not the parameter that defines their efficiency.

Chapter 7

Conclusion

Classification, which is included in the powerful Machine Learning (ML) field, provides a very efficient way to extract information from a huge amount of labeled data. However there is no privacy involved, but in some contexts such as medical data for example, privacy can be compulsory. Yet, designing a privacy-preserving classifier is a challenging quest.

In this thesis, we introduced a generic construction based on both a functional encryption scheme and a classifier. It is generic in the sense that we can use any functional encryption scheme, and also a large variety of classification algorithms in our construction. Regarding cryptography, we experimented it with inner-product functional encryption (IPFE) and also with functional encryption for quadratic polynomial function. Regarding classification, we experimented it with linear classifier, extremely randomized trees and also neural networks. With an IPFE, we got the best performances of classification using neural networks: 96% of classification success on the MNIST test set with 9 secret keys. We got even better results with a quadratic polynomial functional encryption and neural networks: 96% of classification success on the MNIST test set with only 3 secret keys.

We put ourselves in an attacker shoes and studied the security of the construction we proposed. We supposed that an attacker knows which kind of messages are encrypted in our privacy-preserving classifier, and tried to find a way to build a good approximation of original plaintexts with only their IPFE decryptions. For instance, with the MNIST dataset, we consider that an attacker is aware of the digit classification use case. When an IPFE is used, it means that the attacker has some secret keys so for each ciphertexts he has some inner-products between every ciphertexts and every vectors associated with his secret keys. We proposed three attacks: one based on principal component analysis, another one based on fully connected neural networks, and a last one based on convolutional neural networks. We found out that

when 10 IPFE secret keys are provided to someone in a context of MNIST classification, he is able to compute a good approximation for more than 50% of the test set which represent a threat for this use case. We showed that with the quadratic polynomial functional encryption scenario it is easier to classify. We think that an attack on it will be also easier to perform than an attack on the IPFE.

It led us to our deeper study of those vectors associated with the IPFE secret keys. We basically wanted to generate a set of vectors resisting to our attacks, but in the same time that are good-enough for classifying handwritten digit images. We investigated by generating vectors randomly on the one hand, and on the other hand by using a generative adversarial network to generate new vectors. We ended up with the conclusion that if a vector set is good for classifying, it implies that it is easy to attack it with neural networks.

We believe that this neural network approach is a good way for estimating the actual leakage of some use cases based on functional encryption schemes. We also think that it is crucial to estimate it before using such schemes with confidential data. Even though functional encryption schemes are cryptographically secure, one must have some consideration for its use case and what is really revealed to its users. Furthermore, it seems that a functional encryption evaluating polynomials of a certain degree will always leak more than what we need just for classification. A solution would be to use other functional encryption schemes able to compute other functions such as max, min, argmax or argmin, in addition of a polynomial evaluation. Indeed it would enable to perform the whole linear prediction in the decryptions process without revealing any transitional values that makes our plaintext reconstruction possible. It would then be very hard to build a good approximation of an object with this setting. However, designing such schemes remains a huge challenge, as of today, particularly when practicality is a concern.

Chapter 8

Résumé en français

Les algorithmes de classification et plus généralement l'apprentissage automatique ont prouvé être des outils très puissants pour détecter de l'information dans un ensemble de données. Comme nous vivons aujourd'hui dans le re du "big data", ils nous aident à extraire l'information présente dans d'énormes quantités do données. Cependant cela soulève des questions concernant la confidentialité, d'où l'importance de créer des systèmes d'apprentissage automatique garantissant la confidentialité. Cette dernière est l'un des buts de la cryptographie, il semble donc logique de vouloir en tirer profit afin de pouvoir réaliser de l'apprentissage automatique sans rien reveler. Nous réalisons que nous manquons cruellement de solutions efficaces garantissant la confidentialité de nos données. Si cela n'était pas le cas, nous pourrions imaginer que notre serveur email soit capable de filtrer ce que nous jugeons indésirable, sans avoir connaissance de nos courriers. On pourrait peut-être aussi détecter seulement les visages de personnes recherches sur nos vidéos de surveillances qui seraient et resteraient chiffrées. Ce qui serait peut-être encore plus populaire serait que les hôpitaux pourraient partager nos données médicales chiffrées afin que des laboratoires puissent s'en servir pour des études sans trahir la confidentialité de ces données sensibles et légalement protegees. Si seulement nous possédions ce genre de solutions, nous pourrions récupérer un peu de cette vie privée que nous avons renoncé depuis l'arrivée de l'informatique dans nos quotidiens.

La confidentialité des données semble ne semble pas faire bon ménage avec d'autres importantes propriétés telles que la capacité à entrainer un modèle ou à prédire. Cela explique pourquoi il n'est pas trivial de construire des systèmes alliant les deux. Dans cette thèse, on se concentre sur le chiffrement fonctionnel [SW05, BSW11, BCFG17] qui est une récente generalisation de la cryptographie à clef publique. Avec un chiffrement fonctionnel, celui que déploie le système va pouvoir autoriser des serveurs à partiellement

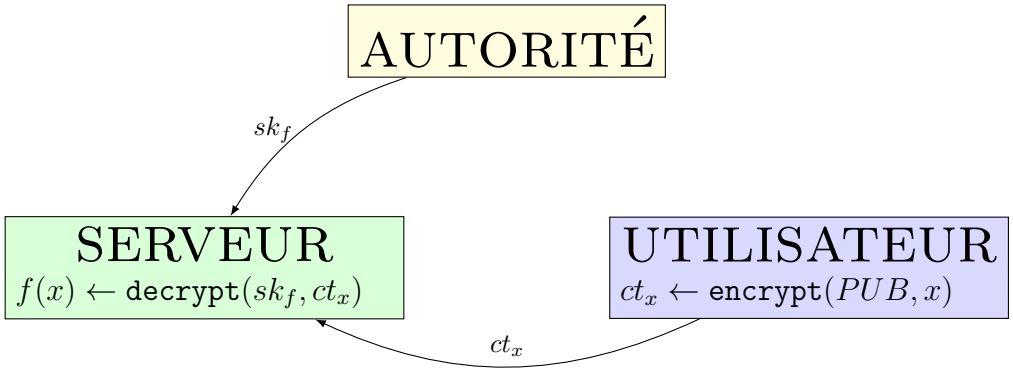


Figure 8.1: Un aperçu des chiffrements fonctionnels.

décrypter des chiffrés, ce qui permet un déchiffrement plus contrôlé offrant une alternative au tout ou rien. L'autorité du système va générer une clef secrète maitresse et une clef publique, et elle va pouvoir générer des clefs secrètes associées à des certaines fonctions grace à sa clef secrète maitresse. Imaginons qu'elle génère une clef secrète associée à la fonction f, qu'elle la donne a un serveur et que ce serveur reçoive un chiffré de m, celui-ci va pouvoir le déchiffrer et utilisant sa clef et obtenir $f(m)$. La figure 8.1 donne un aperçu de ce genre de systèmes.

Dans cette thèse, on propose une construction alliant chiffrements fonctionnels et apprentissage automatique afin de classifier tout en conservant la confidentialité des données fournies en entree. Notre construction est générique dans le sens ou elle peut supporter n'importe quel type de chiffrement fonctionnel. On a voulu faire l'expérience réelle de cette construction, et à l'époque, le seul schéma pratique propose était celui du chiffrement fonctionnel pour le produit scalaire. On a aussi expérimenté ce qui se produirait en supposant que du quadratique existait (ce qui fut le cas un peu plus tard). On a finalement imaginé être des attaquants de notre construction et montré que quand on utilise des chiffrements fonctionnels il faut faire attention à ne pas trop fuiter d'information au risque de perdre la confidentialité une chiffrée.

Apprentissage automatique

Les algorithmes de classement automatique arrivent à identifier à quelle classe un objet appartient. Ce processus est basé sur une phase d'entrainement qui nécessite un ensemble de couples de la forme objet et classe de cet objet.

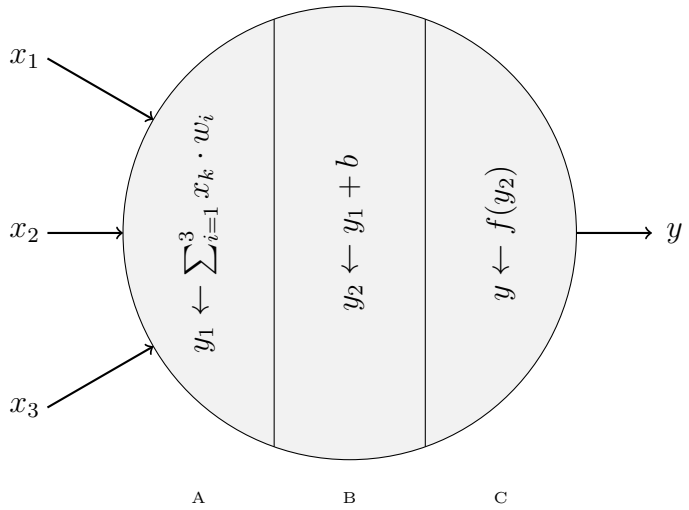


Figure 8.2: Un neurone artificiel avec ses 3 étapes internes : (A) calcul de la somme des poids, (B) ajout du biais, et enfin, (C) calcul de la fonction d'activation f. x_1, x_2 et x_3 sont les valeurs d'entrée et y est la valeur de sortie.

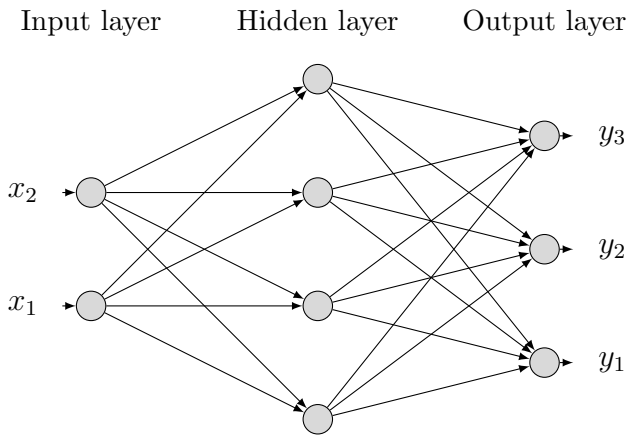
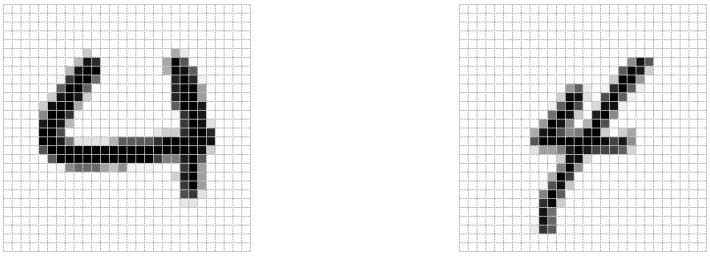


Figure 8.3: L'architecture d'un réseau de neurones.



(a) (b)

Figure 8.4: Exemples d'images de chiffre du MNIST: (a) la 60-ième image (b) la 61-ième image.

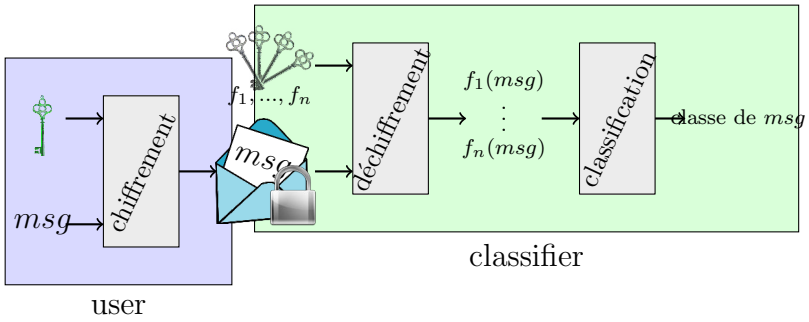


Figure 8.5: Classifieur préservant la confidentialité des données base sur les chiffrements fonctionnels. Le classifieur utilise n clefs secrètes, chacune d'elles permettant l'évaluation d'une fonction de $\{f_i\}_{1 \leq i \leq n}$.

Classification préservant la confidentialité

Avec un classifieur préservant la confidentialité des données, on doit être capable de chiffrer les données de telle façon qu'une autre partie (autorisée) va pouvoir extraire les données nécessaires à sa classification, mais que quiconque autre ne puisse pas inférer quoi que ce soit sur les données chiffrées.

Nous proposons un système de classification préservant la confidentialité des données basé sur des chiffrements fonctionnels et sur des algorithmes de classification. La figure 8.5 illustre notre système ou la confidentialité est obtenue grace au chiffrement fonctionnel, et l'évaluation sur données chiffrées est possible grace aux clefs secrètes du chiffrement fonctionnel.

Dans notre système, il y a une entité appelée *serveur* qui a déjà entraine un modèle de classification sur un ensemble de données d'entrainement. Grâce à cet entrainement, le serveur va pouvoir déterminer les fonctionnalités à évaluer avec le chiffrement fonctionnel dans le but de classifier au mieux.

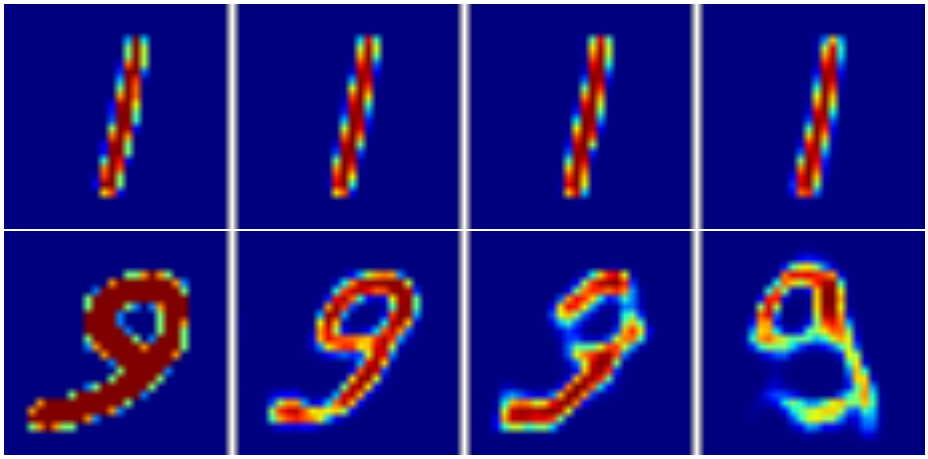


Figure 8.6: Images originales (à gauche) et reconstructions (de gauche à droite: 30, 20 et 10 clefs secrètes) obtenues avec une attaque basée sur des réseaux de neurones convolutionels.

Attaques basées sur la fuite d'information

Nous considérons des attaquants qui respectent le protocole et sont curieux d'inférer plus que ce qu'ils ont. Comme ils disposent de valeurs intermédiaires il est possible de concevoir des méthodes pour par exemple approximer l'input qui est sensée rester confidentielle. Ceci n'est pas capture par le modèle de sécurité des chiffrements fonctionnels. Nous proposons des attaques basées sur des algorithmes d'apprentissage automatique et nous allons nous concentrer sur le cas du chiffrement fonctionnel pour le produit scalaire.

Conclusion

Dans cette thèse nous avons proposé une construction générique basée sur un chiffrement fonctionnel et un classifieur. C'est générique dans le sens qu'il est possible de choisir n'importe quel type de chiffrement fonctionnel mais aussi n'importe quel type de classifieur. Nous avons fait des expérimentations avec des chiffrements fonctionnels pour le produit scalaire mais aussi quadratiques, et concernant la classification, nous avons utilisé des classifieurs linéaires, des forêts d'arbres décisionnels, et aussi des réseaux de neurones. Nous avons eu 96% d'efficacité de classification en utilisant un chiffrement fonctionnel pour le produit scalaire (avec 9 clefs secrètes) et un réseau de neurones. Avec des chiffrements fonctionnels quadratiques, on a la même efficacité en utilisant seulement 3 clefs secrètes.

On s'est aussi mis à la place d'un attaquant et nous avons étudié la sécurité de notre construction. On suppose qu'un attaquant a connaissance de quel genre de messages sont chiffrés dans notre système de classification ayant pour but de conserver la confidentialité des données. Son but est de calculer une approximation des inputs qui ont été chiffrés. Par exemple dans le cas où on utilise la base de données du MNIST qui regroupe des images de chiffres écrits á la main, le but est d'être capable de déterminer quel chiffre est écrit sans avoir connaissance de la façon dont le chiffre a ete tracé. Ceci implique que le but de l'attaquant est d'obtenir le tracé des messages qui ont ete chiffres. Si on imagine que nous avons utilisé un chiffrement fonctionnel pour le produit scalaire, l'attaquant possède un certain nombre de produits scalaires (entre des vecteurs connus et l'image qui doit rester caché) qu'il peut exploiter á sa guise. Nous avons proposé trois attaques : une première qui est basée sur l'analyse en composantes principales, une seconde qui se sert de réseaux de neurones entièrement connectés, et une troisième qui utilise des réseaux de neurones convolutions. Nous avons montré qu'avec 10 clefs secrètes, l'attaquant peut construire une bonne approximation pour plus 50%% du test set, ce qui est une menace non négligeable. Lorsqu'un chiffrement fonctionnel quand pratique est utilisé, il est plus facile de classifier, ce qui nous amèné à penser que la fuite d'information est encore plus grande.

Tout ceci nous a conduits à une étude plus profonde des vecteurs associés aux clefs secrètes du chiffrement fonctionnel pour le produit scalaire. L'idée est de générer des vecteurs pouvant résister à nos attaques. Nous avons pu voir que pour un nombre de vecteurs fixés, l'efficacité des attaques baisse lorsque ces vecteurs ont ete pris aléatoirement, ce qui n'est pas étonnant. Nous avons par la suite utilisé des réseaux antagonistes génératifs afin de générer des vecteurs les plus résistants aux attaques mais en même temps bon pour classifier. Nous sommes arrivé à la conjecture que si un ensemble de vecteurs est bon pour classifier, il est aussi bon pour reconstruire, parce que dans le cas d'usage de la reconnaissance de chiffre, les deux sont intimement liés.

Nous pensons que cette approche basée sur les réseaux de neurones est une bonne façon pour estimer la fuite d'information dans des cas d'utilisation qui se servent de chiffrements fonctionnels. Nous pensons aussi qu'il est très important d'estimer cette fuite avant de déployer un système qui traitera des données confidentielles. Même si le chiffrement fonctionnel en question est prouvé sur, il faut quand même se poser la question de qu'est ce qui peut être appris des données que l'on chiffre.

Dans le cas de la classification, une solution afin de ne rien apprendre de plus sur un input que sa classe, serait d'utiliser des chiffrements fonctionnels

capables d'évaluer des fonctions plus complexes tels que max, min, argmax ou argmin, en plus d'une évaluation polynomiale. En effet, cela permettrait de calculer la totalité de la classification linéaire avec le chiffrement fonctionnel et donc ne reveler aucune valeur intermédiaire (qui sert aux attaques). Concevoir de tels cryptosystèmes reste aujourd'hui quelque chose de difficile.

Bibliography

[ABDCP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In *IACR International Workshop on Public Key Cryptography*, pages 733–751. Springer, 2015.

[ACF+17] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. Cryptology ePrint Archive, Report 2017/972, 2017. http://eprint.iacr.org/2017/972.

[ACG+16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 284–293. ACM, 1997.

[AGH10] Carlos Aguilar-Melchor, Philippe Gaborit, and Javier Herranz. Additively Homomorphic Encryption with d-Operand Multiplications. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 138–154, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[AGRW17] Michel Abdalla, Romain Gay, Mariana Raykova, and Hoeteck Wee. Multi-input inner-product functional encryption from pairings. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*,

pages 601–626, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *Advances in Cryptology–CRYPTO 2013*, pages 500–518. Springer, 2013.

[AH05] Shipra Agrawal and Jayant R Haritsa. A framework for high-accuracy privacy-preserving mining. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 193–204. IEEE, 2005.

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[ans] Recommendations from the french network and information security agency (anssi) about information systems security. `http://www.ssi.gouv.fr/uploads/2015/01/RGS_v-2-0_B1.pdf`

[AP04] Charu C Aggarwal and S Yu Philip. A condensation approach to privacy preserving data mining. In *International Conference on Extending Database Technology*, pages 183–199. Springer, 2004.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 297–314, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[ASP12] Jacob Alperin-Sheriff and Chris Peikert. Circular and kdm security for identity-based encryption. In *International Workshop on Public Key Cryptography*, pages 334–352. Springer, 2012.

[BCD+94] Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Lawrence D Jackel, Yann LeCun, Urs A Muller, Edward Sackinger, Patrice Simard, et al. Comparison of classifier methods: a case study in handwritten digit recognition. In

International conference on pattern recognition, pages 77–77. IEEE Computer Society Press, 1994.

[BCFG17] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In *Annual International Cryptology Conference*, pages 67–98. Springer, 2017.

[BDF18] Guillaume Bonnoron, Léo Ducas, and Max Fillinger. Large FHE gates from tensored homomorphic accumulator. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 217–251, Marrakesh, Morocco, May 7–9, 2018. Springer, Heidelberg, Germany.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology – CRYPTO 2001*, pages 213–229. Springer, 2001.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 2012.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[BKS16] Zvika Brakerski, Ilan Komargodski, and Gil Segev. Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 852–880. Springer, 2016.

[BLLN13] Joppe W Bos, Kristin E Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In *IMA Int. Conf.*, pages 45–64. Springer, 2013.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with er-

rors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584. ACM, 2013.

[BO13] Mihir Bellare and Adam O'Neill. Semantically-secure functional encryption: Possibility results, impossibility results and the quest for a general definition. In *International Conference on Cryptology and Network Security*, pages 218–234. Springer, 2013.

[Bon98] Dan Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*, pages 48–63. Springer, 1998.

[BR15] Jean-François Biasse and Luis Ruiz. FHEW with efficient multibit bootstrapping. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 119–135, Guadalajara, Mexico, August 23–26, 2015. Springer, Heidelberg, Germany.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[BRS13] Dan Boneh, Ananth Raghunathan, and Gil Segev. Function-private identity-based encryption: Hiding the function in functional encryption. In *Advances in Cryptology–CRYPTO 2013*, pages 461–478. Springer, 2013.

[BS15] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In *Theory of Cryptography Conference*, pages 306–324. Springer, 2015.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[C+15] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[CdWM+17] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017:35, 2017.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.

[CM09] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems*, pages 289–296, 2009.

[DH76a] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[DH76b] Whitfield Diffie and Martin E Hellman. Multiuser cryptographic techniques. In *Proceedings of the June 7-10, 1976, national computer conference and exposition*, pages 109–112. ACM, 1976.

[Die00] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[DOT18] Pratish Datta, Tatsuaki Okamoto, and Junichi Tomida. Full-hiding (unbounded) multi-input inner product functional encryption from the k-linear assumption. In *IACR International Workshop on Public Key Cryptography*, pages 245–277. Springer, 2018.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `http://eprint.iacr.org/2012/144`.

[FWP07] Benjamin CM Fung, Ke Wang, and S Yu Philip. Anonymizing classification data for privacy preservation. *IEEE transactions on knowledge and data engineering*, 19(5), 2007.

[G+09] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[Gal12] Steven D Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. volume 63, pages 3–42. Springer, 2006.

[GGG+14] Shafi Goldwasser, S Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 578–602. Springer, 2014.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49. IEEE, 2013.

[GGHZ16a] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In *Theory of Cryptography Conference*, pages 480–511. Springer, 2016.

[GGHZ16b] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 480–511, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany.

[GKP+13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 555–564. ACM, 2013.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and

Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *Advances in Cryptology–CRYPTO 2012*, pages 162–179. Springer, 2012.

[HJP13] W. Hart, F. Johansson, and S. Pancratz. FLINT: Fast Library for Number Theory, 2013. Version 2.4.0, `http://flintlib.org`.

[HSM+00] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000.

[Jol86] Ian T Jolliffe. Principal component analysis and factor analysis. In *Principal component analysis*, pages 115–128. Springer, 1986.

[KLM+16] Sam Kim, Kevin Lewi, Avradip Mandal, Hart William Montgomery, Arnab Roy, and David J Wu. Function-hiding inner product encryption is practical. *IACR Cryptology ePrint Archive*, 2016:440, 2016.

[KS17] Ilan Komargodski and Gil Segev. From minicrypt to obfustopia via private-key functional encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 122–151. Springer, 2017.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology – EUROCRYPT 2008*, pages 146–162. Springer, 2008.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. volume 86, pages 2278–2324. IEEE, 1998.

[LCB] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST Database. http://yann.lecun.com/exdb/mnist/.

[LHH+07] Li Lu, Jinsong Han, Lei Hu, Yunhao Liu, and Lionel M Ni. Dynamic key-updating: Privacy-preserving authentication for rfid systems. In *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*, pages 13–22. IEEE, 2007.

[LL16] Kwangsu Lee and Dong Hoon Lee. Two-input functional encryption for inner products from bilinear maps. *IACR Cryptology ePrint Archive*, 2016:432, 2016.

[LYZ+13] Ming Li, Shucheng Yu, Yao Zheng, Kui Ren, and Wenjing Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *IEEE transactions on parallel and distributed systems*, 24(1):131–143, 2013.

[McE78] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114–116, 1978.

[MMCS11] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 52–59, 2011.

[MTS+12] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. Gupt: privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 349–360. ACM, 2012.

[MWF08] Olvi L. Mangasarian, Edward W. Wild, and Glenn M. Fung. Privacy-preserving classification of vertically partitioned data via random kernels. *ACM Trans. Knowl. Discov. Data*, 2(3):12:1–12:16, October 2008.

[NBJ02] George E Nasr, EA Badr, and C Joun. Cross entropy error function in neural networks: Forecasting gasoline demand. In *FLAIRS Conference*, pages 381–384, 2002.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.

[OPW11] Adam O'Neill, Chris Peikert, and Brent Waters. Bi-deniable public-key encryption. In *Annual Cryptology Conference*, pages 525–542. Springer, 2011.

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 214–231. Springer, 2009.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography Conference*, pages 422–439. Springer, 2012.

[RLKD06] Kui Ren, Wenjing Lou, Kwangjo Kim, and Robert Deng. A novel privacy preserving authentication and access control scheme for pervasive computing environments. *IEEE Transactions on Vehicular technology*, 55(4):1373–1384, 2006.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Pure Math*, volume 20, pages 415–440, 1971.

[SL91] S. Rasoul Safavian and David A. Landgrebe. A survey of decision tree classifier methodology. volume 21, pages 660–674, 1991.

[SS15] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321. ACM, 2015.

[SSW09] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *Theory of Cryptography Conference*, pages 457–473. Springer, 2009.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 457–473. Springer, 2005.

[VDGHV10] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010.

[VLSD+10] Peter Van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter Hartel, and Willem Jonker. Computationally efficient searchable symmetric encryption. In *Workshop on Secure Data Management*, pages 87–100. Springer, 2010.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 114–127. Springer, 2005.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *International Workshop on Public Key Cryptography*, pages 53–70. Springer, 2011.

[WEG87] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[WFHP16] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[WZZ+13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[Yeg09] B Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[YZW05] Zhiqiang Yang, Sheng Zhong, and Rebecca N Wright. Privacy-preserving classification of customer data without loss of accuracy. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 92–102. SIAM, 2005.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Chiffrement fonctionnel appliqué à la classification respectant la confidentialité des données : utilisation pratique, performances et sécurité

Mots clés : sécurité, apprentissage automatique, chiffrement fonctionnel, classification, confidentialité

Résumé : L'apprentissage automatique (en anglais machine learning) ou apprentissage statistique, a prouvé être un ensemble de techniques très puissantes. La classification automatique en particulier, permettant d'identifier efficacement des informations contenues dans des gros ensembles de données. Cependant, cela lève le souci de la confidentialité des données. C'est pour cela que le besoin de créer des algorithmes d'apprentissage automatique capable de garantir la confidentialité a été mis en avant. Cette thèse propose une façon de combiner certains systèmes cryptographiques avec des algorithmes de classification afin d'obtenir un classifieur que veille à la confidentialité. Les systèmes cryptographiques en question sont la famille des chiffrements fonctionnels. Il s'agit d'une généralisation de la cryptographie à clef publique traditionnelle dans laquelle les clefs de déchiffrement sont associées à des fonctions.

Nous avons mené des expérimentations sur cette construction avec un scénario réaliste se servant de la base de données du MNIST composée d'images de digits écrits à la main. Notre système est capable dans ce cas d'utilisation de savoir quel digit est écrit sur une image en ayant seulement un chiffre de l'image.
Nous avons aussi étudié la sécurité de cette construction dans un contexte réaliste. Ceci a révélé des risques quant à l'utilisation des chiffrements fonctionnels en général et pas seulement dans notre cas d'utilisation. Nous avons ensuite proposé une méthode pour négocier (dans notre construction) entre les performances de classification et les risques encourus.

Title: Functional encryption applied to privacy-preserving classification: practical use, performances and security

Keywords: security, privacy, functional encryption, classification, machine learning

Abstract: Machine Learning (ML) algorithms have proven themselves very powerful. Especially classification, enabling to efficiently identify information in large datasets. However, it raises concerns about the privacy of this data. Therefore, it brought to the forefront the challenge of designing machine learning algorithms able to preserve confidentiality.

This thesis proposes a way to combine some cryptographic systems with classification algorithms to achieve privacy preserving classifier. The cryptographic system family in question is the functional encryption one. It is a generalization of the traditional public key encryption in which decryption keys are associated with a function.

We did some experimentations on that combination on realistic scenario using the MNIST dataset of handwritten digit images. Our system is able in this use case to know which digit is written in an encrypted digit image.

We also study its security in this real life scenario. It raises concerns about uses of functional encryption schemes in general and not just in our use case. We then introduce a way to balance in our construction efficiency of the classification and the risks.