



Parcours Mathématiques, Cryptologie, Codage et Applications

Internship Report:

Machine Learning on Encrypted Data: the Consistency Model

Chloé HÉBANT

Supervisor:

David POINTCHEVAL



Academic Supervisor:

Duong-Hieu PHAN



March - July 2017

Contents

1	Internship overview	1
1.1	Introduction	1
1.1.1	Host organization	1
1.1.2	Internship objectives	1
1.1.3	Report structure	2
1.2	Objectives and progress	2
1.2.1	Problem statement	2
1.2.2	Research objectives	3
	Main goal	3
	Other goals	3
1.2.3	Internship progress	4
2	Technical background	5
2.1	Introduction to machine learning	5
2.1.1	Definition	5
2.1.2	Unsupervised learning	6
2.1.3	Supervised learning	7
2.2	Cryptologic background	8
2.2.1	Homomorphic encryption	8
2.2.2	Indistinguishability	9
2.2.3	Cryptographic hardness assumptions	9
2.2.4	Zero-knowledge proofs	11
2.2.5	Non-interactive zero-knowledge proofs	12
3	Consistency model	13
3.1	Presentation of the model	13
3.1.1	The model	13
	Learning phase:	13
	Classification phase:	14
3.1.2	The model with encrypted data	14
3.1.3	Difficulties	16
3.2	Our solution	18
3.2.1	Our protocol	18
	Learning:	19
	Classification:	21

3.2.2	Why does it work?	23
3.3	Formally	26
3.3.1	Zero-knowledge proofs	26
3.3.2	The security games	28
	Sender's privacy:	28
3.4	User's privacy:	34
4	Conclusion and future work	37
4.1	General conclusion	37
4.2	Personal conclusion	37
5	Bonus: works on Matlab	41
5.1	Linear regression	41
5.2	Logistic regression	44
5.3	Multi-class classification	48
5.4	Neural networks	50

Acknowledgements

I respect and thank **Duong-Hieu Phan** and **David Pointcheval**, my supervisors for providing me an opportunity to do this interesting internship at the ENS.

I would especially like to thank Duong-Hieu Phan, for his support throughout the two-year master's degree. I am particularly indebted to the plentiful opportunities that I would never have imagined.

I am extremely grateful to David Pointcheval for having always been available even though he is at the head of the Crypto team and of the Computer department. He transmitted me an incredible amount of advice, and has granted me his time to answer my numerous questions during those 6 months.

I am thankful to all of those with whom I have had the pleasure to work during this internship. Each of **the members of the Crypto Team** has provided me extensive personal and professional guidance and taught me a great deal about scientific research. You warmly welcomed me, I will be glad to see you again for the continuation of my training.

Also, I would like to thank all **the academic team of the CRYPTIS master's degree** without which I would not have been able to discover all this things about cryptology and all its various and enriching sub field.

Internship overview

To support its students, the Cryptis master of Limoges includes in its program an end-of-year internship. I wish to make research in cryptology so I took the specialisation in mathematics, cryptology, coding theory and applications. More precisely, I am interested in privacy. This document presents a summary of the work I have done in this field at the research lab of the ENS in Paris.

1.1 Introduction

In this part, I will first set the scene by beginning with the landscape, the general internship objectives and by presenting the structure of this report. Then, I will detail the subject of study, the objectives we wanted to achieve and an overview in terms of organisation during those 5 months.

1.1.1 Host organization

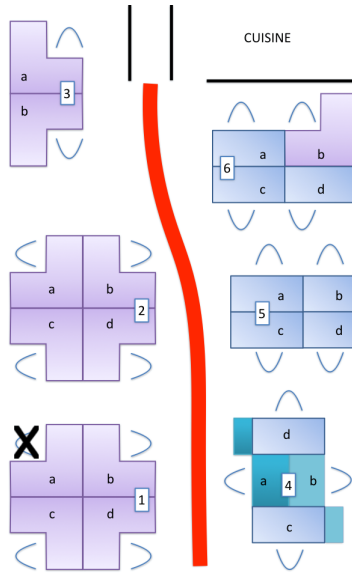
The Ecole Normale Supérieure has been founded in 1794, and is installed in rue d'Ulm since 1874. It is an old school for research and teaching. Since 2010, the ENS is member of Paris Science et Lettres a “federation of 25 prestigious self-governic academic and research institutions” chaired by Marc Mézard.

I made my internship within the Computer Science Department which has been created in 1998 and is now gathering 10 teams. The research laboratory is affiliated with CNRS and INRIA. The Cascade team is one of the four teams in algorithms and analysis section and the cryptology team of the ENS.

The team directed by David Pointcheval is composed of two dozens people: researchers, post-PhD and PhD students. We were two interns in an open space with the PhD students. Different whiteboards on the walls make the collaborative works easier and a meeting room allows the team or the invited people to make presentations, conferences or seminars.

1.1.2 Internship objectives

The first objective of the internship is to make the transition between the first semester of the second-year master degree's during which we attended courses, and the working life of a researcher. I integrated a welcoming team that allowed me to see what the work



of a research lab is like. I can already say that I learned a lot from the PhD working methods to subtleties in proofs or article writing.

This internship addresses issues pertaining to the protection of privacy, to which I would like to contribute. It will pave the way for the thesis I will make during the next three years.

1.1.3 Report structure

As it can be read, the first part of this report is the overview of the internship.

The second part concerns the basic knowledges in machine learning and in cryptology. It gives a simplified background with definitions to be able to understand the next and most important part.

The third part presents my work. After explaining the model I used, I give the protocol we constructed and its security proof.

1.2 Objectives and progress

1.2.1 Problem statement

Why do we want to construct machine learning working on encrypted data?

Because we learned how to solve them, some situations may appear simple for us to treat. The classical example is reading handwritten text. However, we do not always know how to translate from that we learned in terms of traditional algorithms or, sometimes, we just have a partial help and it is not sufficient to automatically treat big data.

Since the end of the 1990s, researchers have been interested by the question: how to make a machine learn? The field of machine learning was born. Since then, lots of learning methods were found and became very efficient, quickly becoming indispensable and inescapable. Now, machine learning is present in a lot of domains such as: autonomous driving, healthcare, music generation, banking or natural language processing.

However, a learning problem uses data as raw material. Our data. We do not want everybody to be able to track our car, we do not want anybody to be able to modify our medical equipment, we do not want the bank to deny a house credit because someone attacked our data before applying a learning algorithm.

This is why there is a urgent need to create solutions providing the benefits without this drawback of machine learning.

1.2.2 Research objectives

To solve the problem, exploration paths use FHE. The purpose of this internship was to find another way without using FHE which would be more efficient.

Main goal

I had successive objectives depending on my success:

1. find a machine learning model which *could* work with encrypted data;
2. find an efficient protocol to make the learning *or* the classification of the previous model;
3. find an efficient protocol to make the learning *and* the classification of the previous model;

We worked on a simple model but the goal number 3 was achieved!

Other goals

Learn a new domain, the machine learning but also establish initial contact with some other subjects thanks to meetings, discussions with the team. Beside the work on my research subject, I discovered in particular:

- the existence of SNARK,
- that there is a world called $i\mathcal{O}$,
- that the FHE could be implemented,
- some other things ...

1.2.3 Internship progress

During my internship, there were different periods, as illustrated on the diagram below. We quickly found the consistency model and we worked on it since March. In the same time, I followed the MOOC on machine learning. In April I worked to prepare Eurocrypt, it was fun to make all the badges and other assets but I stopped working on the consistency model. I participated, of course, to Eurocrypt at the beginning of May. After that we began to have a protocol until mid-July, it had some (important) changes but at the same time, I worked on the article, mainly on the security proof.

Moreover, I attended presentations at the ENS or around each time I had the opportunity. I participated at a week-end of workshops (for which I helped to make the badge design) just before Eurocrypt and at Paris Crypto Day.

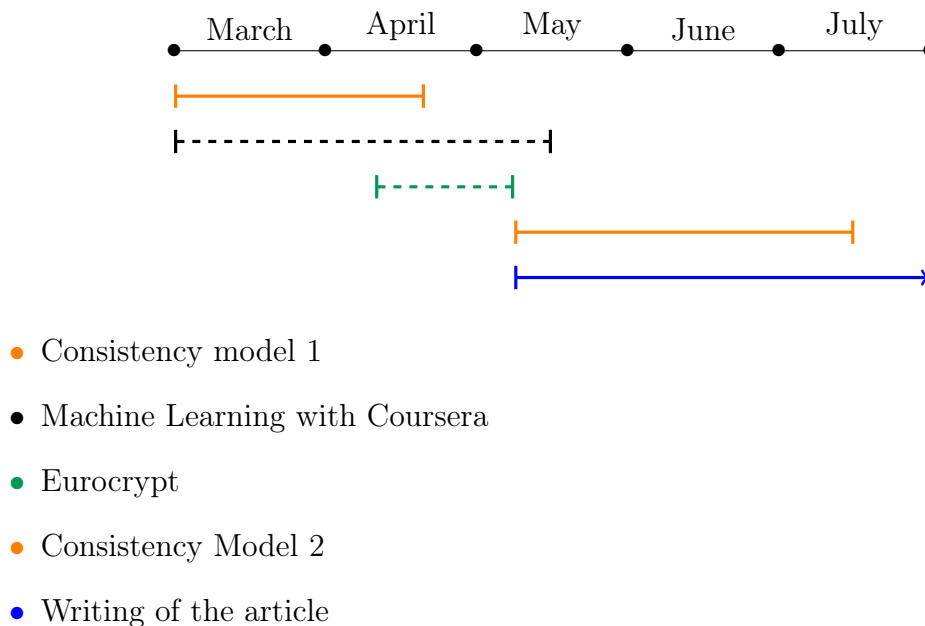


Figure 1.1: Internship progress

Technical background

2.1 Introduction to machine learning

Before talking about cryptology in machine learning, we must know what machine learning is. This part is not here to present an entire course of the subject but just to be aware of the important notions. It will be useful to understand in which framework the work in the next part holds.

At the beginning of the internship, I knew nothing at all on that matter. To study machine learning, I took the course of Andrew Ng, an adjunct professor at Stanford University, and I made the first half of all the exercises and practical works. It provided me a general overview on the subject and allowed me to program some simple schemes with Matlab (that I used for the first time). There is a presentation of what I did in the bonus section at the end of the report (cf part 5). I also read some course notes such as the course of Theoretical Machine Learning by Rob Schapire [Sch14], articles such as an article by Rivest in 1993 about *Cryptography and Machine Learning* [Riv93] and books. I also attended a seminar near the ENS by Guillermo Sapiro (and read his article) named *Learning to Succeed while Teaching to Fail: Privacy in Closed Machine Learning Systems* [SQRS17]. I did not want to be an expert but just to have enough knowledge to be able to make links with cryptography immediately. That is, I wanted to discover the world of machine learning with the eye of someone with cryptology notions. Some precise details, beside the difficulty to understand them, tend toward the possibilities of working on encrypted data.

For example, we do not use the real numbers in cryptology and we do not know how to work with too complex functions. So even if recent functions are found and improved with deep studies to perform learning algorithms, I focused on quite old works.

2.1.1 Definition

Let us begin with the history. The seminar research on machine learning began in 1949 by Arthur Lee Samuel, a pioneer of artificial intelligence research. It is assumed that he coined the term “machine learning”. In 1959, in the IBM Journal of Research and Development, he wrote and gave the first definition:

“Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort”

The definition itself is not really clear but we understand what he meant. This implies that machine learning is a field of study where computer can *learn* without an explicit

programming but with *experience*. What is meant by *learn* is quite general. By *learn* we may want to automatically complete a task, to make accurate predictions or to behave smartly. An example is clearer: Arthur L. Samuel is famous in the field of machine learning for his program for playing draughts that “won a fine game” for Donald Knuth [Knu90] against America’s number-four-ranked player in 1962.

To have a more explicit definition we need to wait 1998 for Tom Mitchell to define what *learn* is by:

“A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E .”

This definition can be illustrated by the figure 2.1 with two phases:

- the first phase called *learning* or *training*: with a training set and a learning algorithm, we obtain a learned function h .
- the second phase called *classification* or *deployment*: with a new example and a learned function h , we obtain a predicted classification.

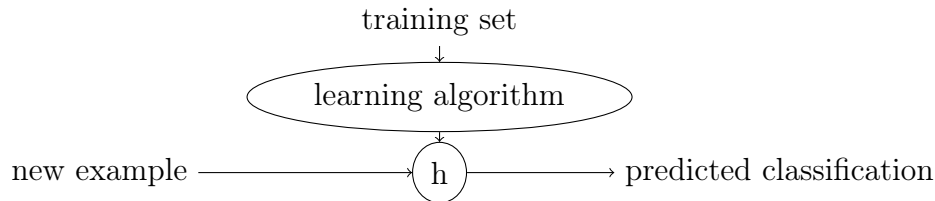


Figure 2.1: General machine learning scheme

The machine learning problems are divided into two groups according to the training set: those that we call unsupervised learning and those that are supervised.

2.1.2 Unsupervised learning

The unsupervised learning is used when we do not have idea of what form the results should have. This type of learning is used to approach problems by deriving structure from data without knowing the effect of the variables. The difficulty is that with unsupervised learning, we do not have feedback on our predictions. So, how is it possible to solve this kind of problems? Variables are often correlated with the data but we do not know exactly with which relationship. The learning algorithm needs to highlight them.

Unsupervised learning are used in a wide variety of applications. For example, it is used in computer vision to recognize objects, in genetic and medical imaging for segmentation.

There is a method called cluster analysis that is the most common method used to create groups or to find hidden patterns in data.

With clustering algorithms, it is possible to partition data into k distinct clusters based on distance to the centroid of a cluster or to use neural networks that learn the topology and distribution of the data.

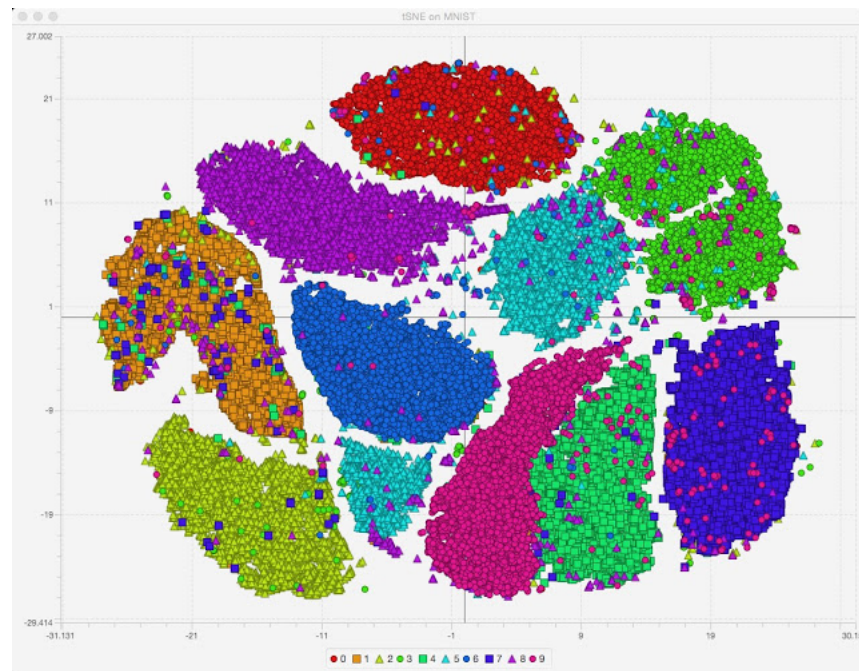


Figure 2.2: Edward Raff's tSNE representation of the MNIST dataset. The MNIST dataset is a dataset of handwritten numbers. It is interesting to remark that there are 9 among the 7s and 7 among the 1s.

2.1.3 Supervised learning

In supervised machine learning, the training set has a special form: each data is labelled (see fig. 2.3). This is very efficient problems where it is easy to label data but explaining why we put this label is difficult. For example, when we see handwritten numbers, it is easy to identify the number but explain why and how we recognize it is difficult.

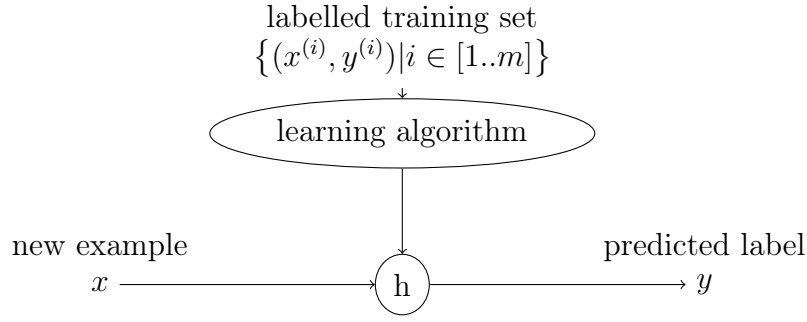


Figure 2.3: Supervised learning scheme

The supervised learning algorithms are divided in two categories:

- the regression learning algorithms: when we try to predict a real value, the program has results within a continuous function. In this case, the learned function maps input variables to some continuous function;
- the classification learning algorithms: when we try to categorize objects into fixed categories, the program has results in a discrete output. In this case, the learned function maps input variables into discrete categories.

In the next part, the learning algorithm presented on encrypted data is a classification learning algorithm.

2.2 Cryptologic background

In this section I begin to talk about cryptology. As in the previous section, I will not make a full course beginning at the very start. I suppose some cryptology notions known to the reader. However I make this part to remind some important definitions that I will use a lot when I will present our protocol.

2.2.1 Homomorphic encryption

Our protocol works thanks to the property of homomorphism:

Definition 1 (Homomorphic Encryption). An encryption E is homomorphic for the operation \odot if it is possible to obtain $E(x \odot y)$ from $E(x)$ and $E(y)$ without decrypting x and y .

Usual operation for \odot is the addition or the multiplication. In our protocol, we use a multiplicative homomorphism.

Definition 2 (Fully Homomorphic Encryption). An encryption E is fully homomorphic if there exists an algorithm **Eval** such that it is possible for any function f to obtain $E(f(x_1, \dots, x_n))$ from $\text{Eval}(E(x_1), \dots, E(x_n))$.

2.2.2 Indistinguishability

With the next definitions we will talk about indistinguishability. It is a security notion. A way to see that an attacker learns nothing about encryptions or to model the fact that an encryption leaks nothing is to prove that an attacker can not decide if a given encryption comes from one message or another even though the adversary can choose the two messages. It exists three notions of indistinguishability: perfect indistinguishability, statistical indistinguishability and computational indistinguishability.

Beginning with the definition of statistical distance, we will formally define the notion of computational indistinguishability. In our protocol we will use this property to prove the sender's privacy (cf 3.3.2).

Definition 3 (Statistical Distance). Let X and Y two random variables in a same finite set \mathcal{X} . The statistical distance between X and Y is defined by:

$$\Delta(X, Y) = \sum_{x \in \mathcal{X}} |\Pr(X = x) - \Pr(Y = x)|$$

With the previous definition, we can define the statistical indistinguishability:

Definition 4 (Statistical Indistinguishability). $X := \{X_n\}_{n \in \mathbb{N}}$ and $Y := \{Y_n\}_{n \in \mathbb{N}}$ are (statistically) indistinguishable means that for all $n \in \mathbb{N}$ the statistical distance between X_n and Y_n is a negligible function of n .

Two random variables are said perfectly indistinguishable when they are statistically indistinguishable and their statistical distance is null.

In most of the cases, the attacker does not have an infinite amount of time. So the computational indistinguishability was created to consider a polynomial-time attacker. Goldreich said in *Foundations of Cryptography* [Gol06] “Objects are considered to be computationally equivalent if they cannot be told apart by any efficient procedure”.

Definition 5 (Polynomial-time Indistinguishability). Let $X := \{X_n\}_{n \in \mathbb{N}}$ and $Y := \{Y_n\}_{n \in \mathbb{N}}$ two probability distributions. X and Y are indistinguishable in polynomial-time if for every probabilistic polynomial-time algorithm, D , every polynomial $p(\cdot)$, and all sufficiently large n 's:

$$\text{Adv}^D := |\Pr(D(X_n, 1^n) = 1) - \Pr(D(Y_n, 1^n) = 1)| < \frac{1}{p(n)}$$

Moreover, D is called distinguisher and Adv^D is its advantage.

2.2.3 Cryptographic hardness assumptions

The security in cryptology lies on assumptions supposed to be hard. We will present two assumptions and define the one we need to our protocol.

Definition 6 (Discrete Logarithm (DL) problem). Let \mathbb{G} be a cyclic group of prime order q and g be a generator of \mathbb{G} . Find a probabilistic polynomial time Turing machine that on input g, g^x outputs x with non-negligible probability.

Definition 7 (DL assumption). There is no probabilistic polynomial time Turing machine that can solve the DL problem on \mathbb{G} .

Definition 8 (Computational Diffie-Hellman (CDH) problem). Let \mathbb{G} be a cyclic group of prime order q and g be a generator of \mathbb{G} . Find a probabilistic polynomial time Turing machine that on input g, g^x, g^y outputs g^{xy} with non-negligible probability.

solving DL problem \Rightarrow solving CDH problem

If there is a probabilistic polynomial time Turing machine that solves the DL problem on \mathbb{G} , there is a probabilistic polynomial time Turing machine that solves the CDH problem on \mathbb{G} . It suffices to compute the discrete logarithm of g^x or g^y and to make an exponentiation.

Definition 9 (CDH assumption). There is no probabilistic polynomial time Turing machine that can solve the CDH problem on \mathbb{G} .

Definition 10 (Decisional Diffie-Hellman (DDH) problem). Let \mathbb{G} be a cyclic group of prime order q and g be a generator of \mathbb{G} . Let the two distributions:

- $\mathcal{X} : g, g^x, g^y$ and g^{xy} , where $x, y \in \mathbb{Z}_q$, are random scalars chosen uniformly at random;
- $\mathcal{Y} : g, g^x, g^y$ and g^z , where $x, y, z \in \mathbb{Z}_q$, are random scalars chosen uniformly at random;

Find a probabilistic polynomial time Turing machine that on input g, g^x, g^y, g^z can correctly decide with non-negligible probability if the quadruple comes from the distribution \mathcal{X} or \mathcal{Y} .

solving CDH problem \Rightarrow solving DDH problem

If there is a probabilistic polynomial time Turing machine that solves the CDH problem on \mathbb{G} , there is a probabilistic polynomial time Turing machine that solves the DDH problem on \mathbb{G} . It suffices to compute g^{xy} and to compare it with g^z .

Definition 11 (DDH assumption). There is no probabilistic polynomial time Turing machine that solves the DDH problem on \mathbb{G} .

There exist some other cryptographic hardness assumptions but we do not use them so I do not present them. In our protocol we will prove a certain form of indistinguishability by using the DDH assumption. Roughly speaking, we will have (g^α, h^β) with $h = g^x$. An attacker can not know if β is equal to α or not and so, he can not know if it is the encryption of a 0 or a 1.

2.2.4 Zero-knowledge proofs

In our protocol, we will need to prove some relations between elements in groups. For that, we will use zero-knowledge proofs that we will define here. We will also give some properties they need to satisfy.

A zero-knowledge (ZK) proof is a protocol between two players: a prover and a verifier where the prover needs to convince the verifier of a given statement without revealing any information beyond other than the truth of this statement. They were invented by Goldwasser, Micali and Rackoff in [GMR85].

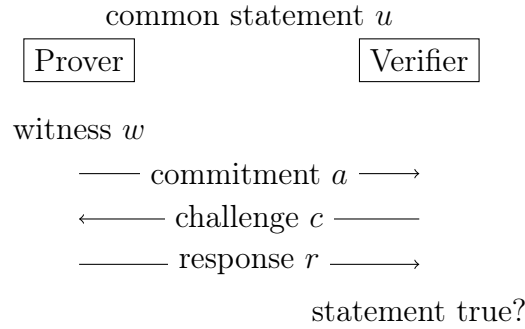


Figure 2.4: ZK protocol

Definition 12. A protocol between two players of the form given in figure 2.4 is a zero-knowledge proof if it satisfies the three properties:

- Completeness: if the prover and the verifier follow the protocol, the verifier always accepts;
- Soundness: a false proof is rejected with probability at least $1/2$;
- Zero-knowledge: whatever the verifier learns, he could have learned by himself without any interaction with the prover.

In our protocol we will use the property of *special soundness* and *special honest-verifier zero-knowledge*:

Definition 13. A zero-knowledge proof verifies the property of special soundness if there exists a probabilistic polynomial-time algorithm E (such as extractor) which given any correct common statement u and any pair of accepting conversation (a, c, r) and (a, c', r') with $c \neq c'$ computes a witness w such that the statement is true.

Definition 14. A zero-knowledge proof verifies the property of special honest-verifier zero-knowledge if there exists a probabilistic polynomial-time algorithm S (such as simulator) which given any statement u and any challenge c produces conversations (a, c, r) with the same probability distribution as conversations between an honest prover using a witness w and an honest verifier on common statement u and challenge c .

There are two kind of zero-knowledge properties: with honest-verifier or not. For our protocol, our verifier are considering honest.

2.2.5 Non-interactive zero-knowledge proofs

In a zero-knowledge proof, it is important that the verifier chooses a random challenge. To obtain a non-interactive zero-knowledge proof, we need to replace this step by the commit of a value. This is know as the Fiat-Shamir heuristic [FS87]. For efficiency and practicality, our protocol uses non-interactive ZK proof and to commit the value we use an hash function.

Definition 15 (Hash Function). An hash function \mathcal{H} is a function with a fixed length output space and the following properties:

- there is an efficient (in time and space) algorithm to compute the function;
- One way: given $y = \mathcal{H}(x)$ it is hard to find x .
- Pre-image resistance: given $y = \mathcal{H}(x)$ it is hard to find x' such that $\mathcal{H}(x') = y$;
- Collision resistance: it is hard to find x, x' such that $\mathcal{H}(x) = \mathcal{H}(x')$.

Thanks to the properties of the hash functions, the prover can not cheat. In fact, the challenge obtained with the hash function has the same form than a verifier's challenge.

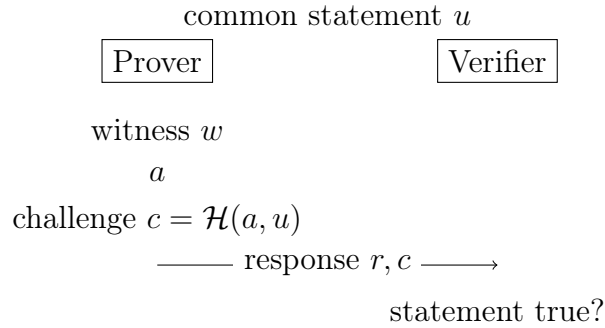


Figure 2.5: Non-Interactive ZK protocol

Consistency model

In this part, I will begin by the description of the model that we used and how to adapt it in a secure version. Then, I will present our protocol with its formal proofs of security.

3.1 Presentation of the model

3.1.1 The model

Learning phase:

Let us consider that we want to learn which combination of variables would cause a specific effect E . A way to do this is to start by filling a database line by line giving at each time a set of variables and the effect this set made. Then, take the subset of lines with label E and check for each variable if it was in a same state on all the lines or rather, if it never appears. The learned formula corresponds to an hypothesis reflecting the data in the base.

For example, we can run n programs π_1 to π_n . Some programs when they are working together, create problems. We want to identify what is the troublesome combination. For that, we fill a database with in a line (x, y) , $x \in \{0, 1\}^n$ and $y \in \mathcal{E} := \{+, -\}$. The j -th bit of x corresponds to the state of the π_j program. The table 3.1.1 below shows an example of the data in this case.

Data					Label
π_1	π_2	π_3	π_4	π_5	
1	1	0	0	1	+
1	0	1	0	0	−
0	1	0	0	1	+
1	1	0	1	1	+
0	1	1	1	1	−

Table 3.1: The data in the database: If a program π_j is running at the time i , we have a 1 at the i th-row and j th-column. If the program is not running then we have a 0. The label indicate if during the situation a problem occurred (symbolized by a positive label) or not (symbolized by a negative label).

To consider the state $E := +$, we take the subset of lines with a positive label and for each column we examine if the program π_j was working or not, i.e. if the j -th column takes only one value 1, one value 0, or both values 0 and 1. With that, we obtain the possible formula that makes the program fail. We have learned a boolean formula from the data. The table 3.1.1 below shows the learning phase.

Data					Label
π_1	π_2	π_3	π_4	π_5	
1	1	0	0	1	+
0	1	0	0	1	+
1	1	0	1	1	+
<div style="display: flex; justify-content: space-around; align-items: center;"> <div>↓</div> <div>↓</div> <div>↓</div> </div>					
★	1	0	★	1	

Table 3.2: Learning the data: with this data, we learn the formula $F(x_1, \dots, x_5) = x_2 \wedge \bar{x}_3 \wedge x_5$

Classification phase:

In the second phase, we have a combination of variables and we wish to have an hypothesis on its effect. For that, we will use the learned formula in the first phase to evaluate the combination.

In our example, we can use $F(x)$ to check if it is possible to have an error with our situation w . We look at $F(w)$. If the answer is $+$, we will have a problem for sure but if the answer is $-$, we can have an error or not.

Example 1. With $F(x_1, \dots, x_5) = x_2 \wedge \bar{x}_3 \wedge x_5$, we have:

$$F(01101) = 0$$

$$F(11001) = 1$$

$$F(01111) = 1$$

Remark 1. To obtain more information, we can calculate an F' with the negative labels and look $F'(w)$. Unfortunately, this situation can end up without having enough information about $F(w)$ and $F'(w)$. In this case, we can not know if w fails or not.

3.1.2 The model with encrypted data

Before working with encrypted data, we can look into the consistency model (see 3.1) with three roles:

- The sender sends data to the database.
- The database executes the learning.

- The user has a word, receives the formula and calculates the hypothetical label.

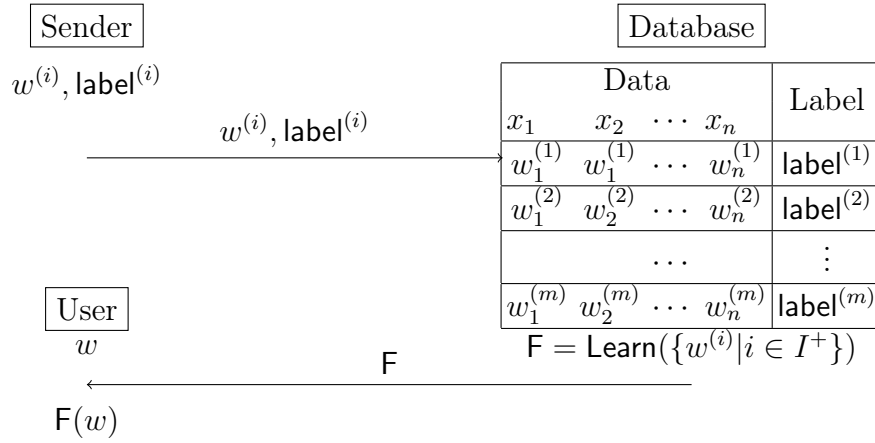


Figure 3.1: Scheme

Now, we want to adapt this scheme in a secure version. That is, we want a scheme in which the sender encrypts his data before sending them to the database so that the data in the database are protected from the others.

We can already see that there is a problem to be solved in obtaining a scheme able to work with encrypted data. Here the user makes the classification by itself. This is good for the user's privacy: nobody can learn his data. In a first approach, the sender just needs to send encrypted data to the encrypted database that performs the learning, the user decrypts the formula and makes all the classification of his choices. Nevertheless, the user has too much power in this situation. Because in the consistency model, having the formula already means having information about the data. To respect the sender's privacy, no one shall have access to the plain formula.

A fourth role is added to solve the problem. The user can not decrypt the formula and to curb him, each time he wants to classify a word, he makes a request to a tester.

But for the same reason as above, the tester must not know or learn the plain formula. In some ways, the database shall not be encoded the same way as the requests. And, as the user does not make his classification himself any more, a new security notion appears. We want a user's privacy: nobody, even the tester, must learn the word the user wants to classify by a request.

The four roles are:

- the sender \mathcal{S} sends vector to the encrypted database **EDB** to fill it;
- the encrypted *public* database **EDB** stores the data and learns the encrypted *public* formula;
- the user \mathcal{U} has a vector and wants to know if his vector verifies the formula by making a request;

- the tester \mathcal{T} helps the user in his queries.

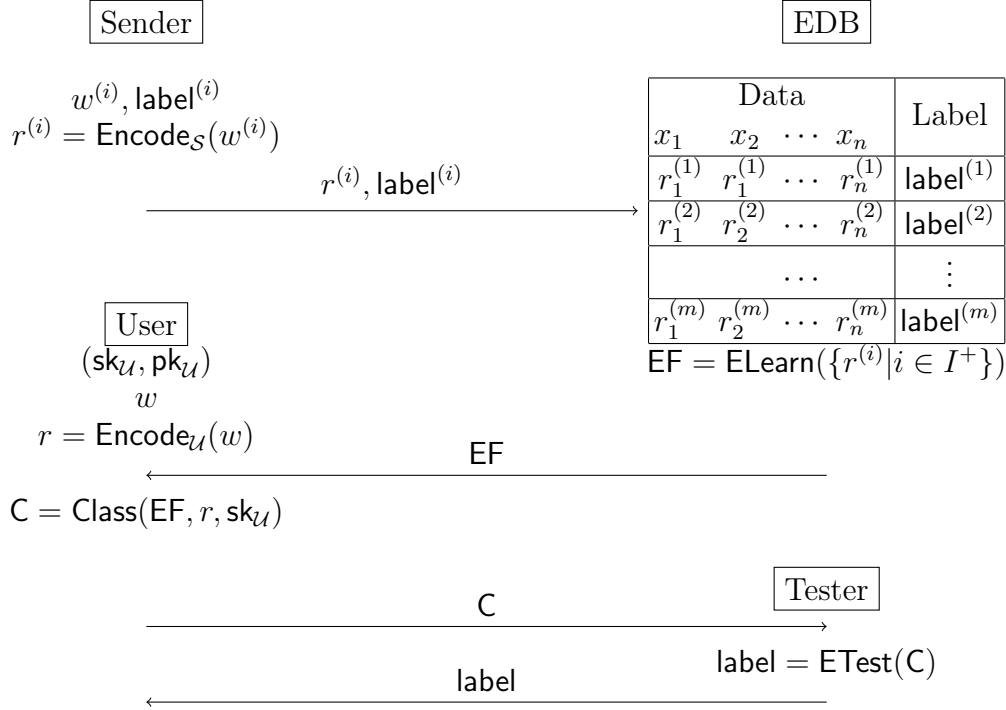


Figure 3.2: Scheme with encrypted data

3.1.3 Difficulties

We have just seen the model. It is very simple and certainly the simplest possible model. It comes at the first chapter in books about machine learning. For human or machines, this learning is easy to solve. Moreover, this model has the particularity and the default not to include errors. When an object is classified positively it is like it is 100% classified positively and 0% negatively. In most cases the learning algorithm outputs an answer with a percentage of credibility. So, why do we study it? *Why is this model interesting?* For us, a non-negligible advantage is that this model works with integer and in particular, it can work with bits. Another important point, there is no suitable solution for machine learning on encrypted data. With a simple model, we can hope to find a sufficient solution or at least a simple solution. However, beyond the consistency model, this study raises some questions valid for a lot of models. The consistency model should not be seen as a model to solve with encrypted data but must be seen as a first step in this field.

Now that the consistency model is raised, *how to find a protocol?* I was a little apprehensive about this question. It requires to put the knowledges and the tools from cryptology into perspective. In this part, I will present a way to see the consistency model. This was our approach but there are probably others. For the rest of this

rapport we will take the example in the part 3.1.1. It will be clearer. Even if we did not have the protocol, the use of homomorphic encryption appeared to be the raw material for our construction. I will explain why.

First, we wanted to focus on the learning phase. The formula we wanted to learn consist in a list of 0, 1 and \star if \star means a mix of 0 and 1 like in 3.1.1.

If the plain database is small, a human can just look at each column and see if there is only 0, only 1 or both by storing the first element, looking the second element, comparing them and repeating the task for the rest of the database. A way to do that in an encrypted database is, with an homomorphic encryption, to use the property of the identity element.

We can encrypt the database in two ways. The first time so that the encryption of 0 is the encryption of the identity element and the encryption of 1 is the encryption of a random element. The second time we make the opposite, the encryption of 1 is the encryption of the identity element and the encryption of 0 is the encryption of a random element.

Then, we can multiply (if the homomorphic encryption is multiplicative otherwise we can add) all the elements in the same column for each encryption. In the first encryption, if the encrypted element is the encryption of the identity element, we know that we have just 0s in the column, if not we look if in the second encryption we have the encryption of the identity element, if yes, we know that we have just 1s in the column, otherwise we say we learn \star .

We solved the learning problem on encrypted data! It was easy but we still have the problem of the previous section: we can not decrypt the learned formula. And with that, we also need to solve the classification problem otherwise the consistency model is meaningless.

Now, we want to find a way to encrypt the user request in a way that a tester can answer the right classification. This classification is linked to the partial computations by the relation:

$$F(w) = 1 \Leftrightarrow \forall j, F_j = w_j \text{ or } \star$$

For a formula with just 0s and 1s, there is an easy method to do that: if the absorbing property is preserved for each column after the multiplication of the encrypted user's element we can accept. Otherwise, we reject it.

However, the problem is more difficult for a formula with \star s because we can not apply this method. In fact, there is no property to conserve before the multiplication of an encrypted user's element. So, after the multiplication of the encrypted user's element we will always reject whereas we want the opposite: we want to accept.

In fact, the formula has two “parts” working in opposite way that we can see in the table below.

F_j	0		1		\star	
w_j	0	1	0	1	0	1
$F_j = w_j$	✓	×	×	✓	×	×
Expected partial classification	✓	×	×	✓	✓	✓

Table 3.3: One difficulty in the consistency model: we want to reject the blue crosses but accept the red crosses

This difficulty lies in the consistency model but there is another difficulty more general to machine learning on encrypted data. The tester must not create his own classification. Hence, the user can not directly send his encoded word and let the tester make the classification with the encoded formula which is public. We need to find a way to mix the data so that a user alone can not extract information from the database, a tester alone can not as well, but so that the learned formula mixed with the user's word gives a request that can be understood by a tester. We need to create all the secrets and to construct all the encryption to obtain a consistent protocol. Last but not least, it is difficult to have a protocol that can be proved secure.

3.2 Our solution

3.2.1 Our protocol

In this subsection, we will detail our protocol. We exploit the property of homomorphism to learn and to classify the data by using El Gamal encryptions. For security reasons, we need to prove that the constructed elements are well formed each time. This introduces zero-knowledge proofs.

For security, we consider that the sender needs to authenticate himself to the database to be able to send data.

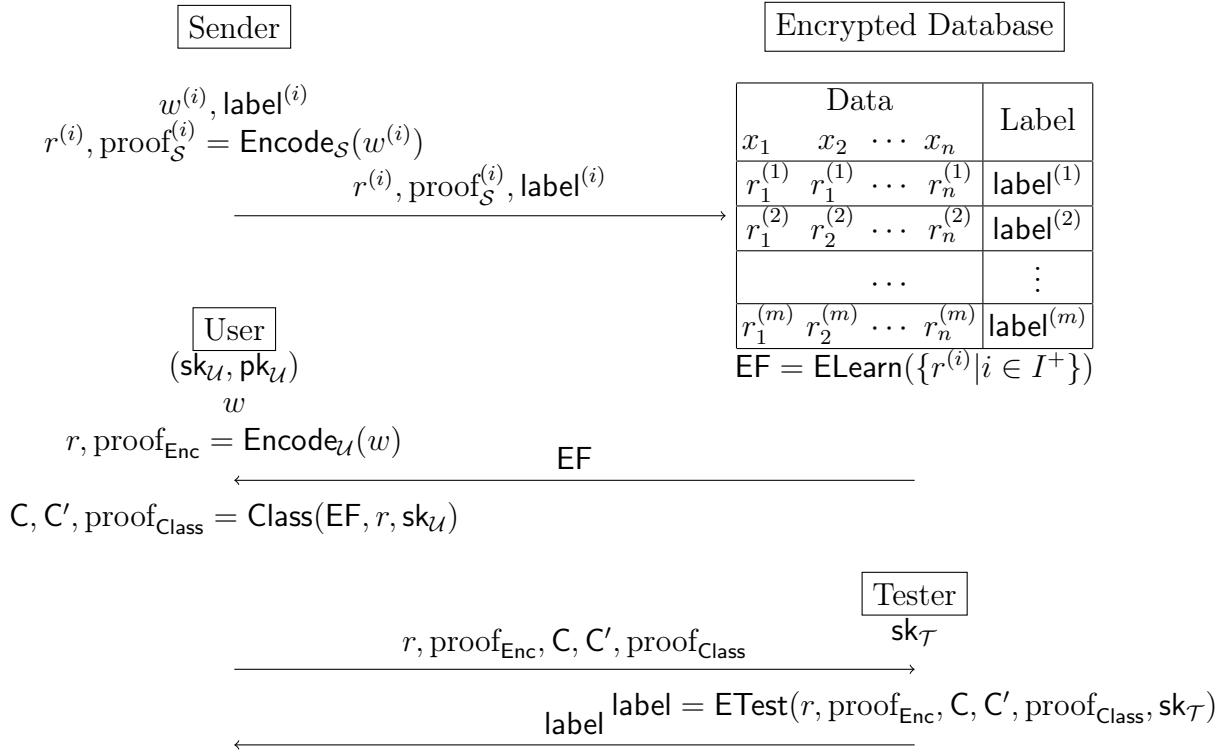


Figure 3.3: Scheme of our protocol

Learning:

- **Setup(1^λ):** Given a security parameter λ , it generates $s \xleftarrow{R} \mathbb{Z}_q$ the master secret key and the parameters $\text{param} = (\mathbb{G}, g, h_0 = g^s, h_1)$ where \mathbb{G} is a group of order q and of generator g and $h_1 \xleftarrow{R} \mathbb{G}$;
- **Encode $_S(\text{param}, w)$:** Given a word $w = (w_1, \dots, w_n)$, it encodes in a sender's form w by creating for each $j \in \{1, \dots, n\}$:

$$r_j := \begin{cases} (g^{\alpha_j}, h_0^{\alpha_j}, h_j) & \text{if } w_j = 1 \\ (g^{\alpha_j}, h_j, h_0^{\alpha_j}) & \text{if } w_j = 0 \end{cases} \quad \text{with } \alpha_j \xleftarrow{R} \mathbb{Z}_q, h_j \xleftarrow{R} \mathbb{G}$$

it makes $\text{proof}_S \leftarrow \text{Prove}_S(\text{param}, w, r, \{\alpha_j\}_j)$

and outputs $(\{r_j\}_j, \text{proof}_S)$;

Each time a sender encodes a word, we need to be able to verify that he makes the right encoding: with g, h_0 and the first element, either the second or the third element

of r_j is the last element of a Diffie-Hellman tuple. Given **param** and r , we want that:

$$\forall j \in \{1, \dots, n\}, \exists \alpha_j : \\ (r_j(1) = g^{\alpha_j}) \wedge ((r_j(2) = h_0^{\alpha_j}) \vee (r_j(3) = h_0^{\alpha_j})) \wedge (r_j(2) \neq r_j(3))$$

Below, we describe how the sender builds his proof and how the verification is made.

- **Prove_S(param, w , r , $\{\alpha_j\}_j$):** Given the encoded word r of w with $\{\alpha_j\}_j$, it proves the relation:

$$\left\{ (r, g, h_0, \{\alpha_j\}_j) : \bigwedge_{j \in \{1, \dots, n\}} (r_j(1) = g^{\alpha_j} \wedge (r_j(2) = h_0^{\alpha_j} \vee r_j(3) = h_0^{\alpha_j})) \right\} \quad (3.1)$$

For each j , let $t_j \in_R \mathbb{Z}_q$

$$A_j \leftarrow g^{t_j}$$

$$\begin{array}{l|l} \text{if } w = 1: \text{ let } c_{j1}, a_{j1} \in_R \mathbb{Z}_q & \text{if } w = 0: \text{ let } c_{j0}, a_{j0} \in_R \mathbb{Z}_q \\ B_j \leftarrow h_0^{t_j} & B_j \leftarrow h_0^{a_{j0}} \cdot r_j(2)^{c_{j0}} \\ C_j \leftarrow h_0^{a_{j1}} \cdot r_j(3)^{c_{j1}} & C_j \leftarrow h_0^{t_j} \end{array}$$

$$c \leftarrow \mathcal{H}(\{A_j\}_j, \{B_j\}_j, \{C_j\}_j)$$

$$a_j \leftarrow t_j - c\alpha_j$$

$$\begin{array}{l|l} c_{j0} \leftarrow c - c_{j1} & c_{j1} \leftarrow c - c_{j0} \\ a_{j0} \leftarrow t_j - c_{j0}\alpha_j & a_{j1} \leftarrow t_j - c_{j1}\alpha_j \end{array}$$

and outputs $\text{proof}_S := (c, \{c_{j0}\}_j, \{a_j\}_j, \{a_{j0}\}_j, \{a_{j1}\}_j)$;

- **Verif_S(param, r , proof_S):** it creates for each j :

$$\begin{array}{l} c_{j1} \leftarrow c - c_{j0} \\ A_j \leftarrow g^{a_j} r_j(1)^c \\ B_j \leftarrow h_0^{a_{j1}} r_j(2)^{c_{j1}} \\ C_j \leftarrow h_0^{a_{j0}} r_j(3)^{c_{j0}} \end{array}$$

and checks if $c = \mathcal{H}(\{A_j\}_j, \{B_j\}_j, \{C_j\}_j)$ and if for all j , $r_j(2) \neq r_j(3)$.

- **ELearn($\{r^{(i)} | i \in I^+\}$):** Given a list of encoded words $\{r^{(i)} | i \in I^+\}$, it computes the learned formula:

$$\text{EF}_j := \left(\prod_{i \in I^+} r_j^{(i)}(1), \prod_{i \in I^+} r_j^{(i)}(2), \prod_{i \in I^+} r_j^{(i)}(3) \right)$$

and outputs $\text{EF} = \{\text{EF}_j\}_j$;

$$\left\{ (r, g, h_0, h_1, \{\alpha_j\}_j) : \bigwedge_{j \in \{1, \dots, n\}} \left((r_j(1) = g^{\alpha_j}) \wedge \left[((r_j(2) = h_0^{\alpha_j}) \wedge (r_j(4) = h_1^{\alpha_j})) \vee ((r_j(3) = h_0^{\alpha_j}) \wedge (r_j(5) = h_1^{\alpha_j})) \right] \right) \right\} \quad (3.2)$$

Classification:

- **KeyGen_U(param):** Given parameters **param**, it generates ($\mathbf{sk}_U = u, \mathbf{pk}_U = h_0^u$) the user's secret and public keys with $u \xleftarrow{R} \mathbb{Z}_q$;
- **KeyGen_T(param):** Given parameters **param**, it generates $\mathbf{sk}_T = su$ the tester's secret key;
- **Encode_U(param, w):** Given a word $w = (w_1, \dots, w_n)$, it encodes w in a user's form by creating for each $j \in \{1, \dots, n\}$:

$$r_j := \begin{cases} (g^{\alpha_j}, h_0^{\alpha_j}, h_j, h_1^{\alpha_j}, h'_j) & \text{if } w_j = 0 \\ (g^{\alpha_j}, h_j, h_0^{\alpha_j}, h'_j, h_1^{\alpha_j}) & \text{if } w_j = 1 \end{cases} \quad \text{with } \alpha_j \xleftarrow{R} \mathbb{Z}_q, h_j, h'_j \xleftarrow{R} \mathbb{G}$$

and $\text{proof}_{\text{Enc}} = \text{Prove}_{\text{Encode}_U}(\text{param}, w, r, \{\alpha_j\}_j)$

It outputs $(\{r_j\}_j, \text{proof}_{\text{Enc}})$;

Again, each time a user encodes a word, we need to be able to verify that he makes the right encoding: with g, h_0 and the first element, either the second or the third element of r_j is the last element of a Diffie-Hellman tuple. Given **param** and r , we want that:

$\forall j \in \{1, \dots, n\}, \exists \alpha_j :$

$$(r_j(1) = g^{\alpha_j}) \wedge \left[((r_j(2) = h_0^{\alpha_j}) \wedge (r_j(4) = h_1^{\alpha_j})) \vee ((r_j(3) = h_0^{\alpha_j}) \wedge (r_j(5) = h_1^{\alpha_j})) \right] \\ \wedge (r_j(2) \neq r_j(3)) \wedge (r_j(4) \neq r_j(5))$$

Below, we describe how we make the proof and perform the verification.

- **Prove_{Encode_U}(param, w, r, {α_j}_j):** Given a encoded word $r = \text{Encode}_U(\text{param}, w)$ and $\{\alpha_j\}_j$, it creates the proof:

For each j , let $t_j \in_R \mathbb{Z}_q$
 $A_j \leftarrow g^{t_j}$

$$\begin{array}{l|l}
\text{if } w_j = 0, \text{ let } c_{j1}, a_{j1} \in_R \mathbb{Z}_q & \text{if } w_j = 1, \text{ let } c_{j0}, a_{j0} \in_R \mathbb{Z}_q \\
B_j \leftarrow h_0^{t_j} & B_j \leftarrow h_0^{a_{j0}} \cdot r_j(2)^{c_{j0}} \\
C_j \leftarrow h_0^{a_{j1}} \cdot r_j(3)^{c_{j1}} & C_j \leftarrow h_0^{t_j} \\
D_j \leftarrow h_1^{t_j} & D_j \leftarrow h_1^{a_{j0}} \cdot r_j(4)^{c_{j0}} \\
E_j \leftarrow h_1^{a_{j1}} \cdot r_j(5)^{c_{j1}} & E_j \leftarrow h_1^{t_j} \\
c \leftarrow \mathcal{H}(\{A_j\}_j, \{B_j\}_j, \{C_j\}_j, \{D_j\}_j, \{E_j\}_j) & \\
a_j \leftarrow t_j - c\alpha_j & \\
c_{j0} \leftarrow c - c_{j1} & c_{j1} \leftarrow c - c_{j0} \\
a_{j0} \leftarrow t_j - c_{j0}\alpha_j & a_{j1} \leftarrow t_j - c_{j1}\alpha_j
\end{array}$$

and outputs $(c, \{c_{j0}\}_j, \{a_j\}_j, \{a_{j0}\}_j, \{a_{j1}\}_j)$;

- $\text{Verif}_{\text{Encode}_{\mathcal{U}}}(\text{param}, r, \text{proof}_{\text{Enc}})$: it creates for each j :

$$\begin{array}{l}
c_{j1} \leftarrow c - c_{j0} \\
A_j \leftarrow g^{a_j} r_j(1)^c \\
B_j \leftarrow h_0^{a_{j1}} r_j(2)^{c_{j1}} \\
C_j \leftarrow h_0^{a_{j0}} r_j(3)^{c_{j0}} \\
D_j \leftarrow h_1^{a_{j1}} r_j(4)^{c_{j1}} \\
E_j \leftarrow h_1^{a_{j0}} r_j(5)^{c_{j0}}
\end{array}$$

and checks if: $c = \mathcal{H}(\{A_j\}_j, \{B_j\}_j, \{C_j\}_j, \{D_j\}_j, \{E_j\}_j)$ and if for all j , $r_j(2) \neq r_j(3)$ and $r_j(4) \neq r_j(5)$.

- $\text{Class}(\text{EF}, r, \text{sk}_{\mathcal{U}})$: Given a encoded formula EF , an encoded word $r = \text{Encode}_{\mathcal{U}}(\text{param}, w)(1)$ and a secret key $\text{sk}_{\mathcal{U}}$, it creates $\mathbf{C} = \{C_j\}_j$:

$$\mathbf{C}_j := (\text{EF}_j(1) \cdot r_j(1), \text{EF}_j(2) \cdot r_j(2), \text{EF}_j(3) \cdot r_j(3))$$

It creates $\mathbf{C}' = \{C'_j\}_j$:

$$\mathbf{C}'_j := (\mathbf{C}_j(1), \mathbf{C}_j(2)^{\text{sk}_{\mathcal{U}}}, \mathbf{C}_j(3)^{\text{sk}_{\mathcal{U}}})$$

and $\text{proof}_{\text{Class}} = \text{Prove}_{\text{Class}}(\text{EF}, r, \mathbf{C}, \mathbf{C}', \text{sk}_{\mathcal{U}})$

It outputs $(\mathbf{C}, \mathbf{C}', \text{proof}_{\text{Class}})$;

This time, we need to be able to check if the classification was made correctly. Given $\text{EF}, r, \mathbf{C}, \mathbf{C}'$, we want that:

$$\begin{aligned}
& \forall j \in \{1, \dots, n\}, \exists \text{sk}_{\mathcal{U}} : \\
& (\mathbf{C}_j(1) = \text{EF}_j(1) \cdot r_j(1)) \wedge (\mathbf{C}_j(2) = \text{EF}_j(2) \cdot r_j(2)) \wedge (\mathbf{C}_j(3) = \text{EF}_j(3) \cdot r_j(3)) \\
& \quad \wedge (\mathbf{C}'_j(1) = \mathbf{C}_j(1)) \wedge (\mathbf{C}'_j(2) = \mathbf{C}_j(2)^{\text{sk}_{\mathcal{U}}}) \wedge (\mathbf{C}'_j(3) = \mathbf{C}_j(3)^{\text{sk}_{\mathcal{U}}})
\end{aligned}$$

Below, we describe how we make the proof and perform the verification.

- $\text{Prove}_{\text{Class}}(\text{EF}, r, \text{C}, \text{C}', \text{sk}_{\mathcal{U}})$: It creates the proof:

$$\left\{ (\text{C}, \text{C}', \text{sk}_{\mathcal{U}}) : \bigwedge_{j \in \{1, \dots, n\}} \left((\text{C}'_j(2) = \text{C}_j(2)^{\text{sk}_{\mathcal{U}}}) \wedge (\text{C}'_j(3) = \text{C}_j(3)^{\text{sk}_{\mathcal{U}}}) \right) \right\} \quad (3.3)$$

For each j , let $t_j \in_R \mathbb{Z}_q$
 $A_j \leftarrow \text{C}_j(2)^{t_j}$
 $B_j \leftarrow \text{C}_j(3)^{t_j}$
 $c \leftarrow \mathcal{H}(\{A_j\}_j, \{B_j\}_j)$
 $a_j \leftarrow t_j - c \text{sk}_{\mathcal{U}}$

It outputs $\text{proof}_{\text{Class}} := (c, \{a_j\}_j)$;

- $\text{Verif}_{\text{Class}}(\text{EF}, r, \text{C}, \text{C}', \text{proof}_{\text{Class}})$: it creates for each j :

$$\begin{aligned} A_j &\leftarrow \text{C}_j(2)^{a_j} \text{C}'_j(2)^c \\ B_j &\leftarrow \text{C}_j(3)^{a_j} \text{C}'_j(3)^c \end{aligned}$$

And checks if: $c = \mathcal{H}(\{A_j\}_j, \{B_j\}_j)$ and if for each j :

$$\begin{aligned} \text{C}_j(1) &= \text{EF}_j(1) \cdot r_j(1) \\ \text{C}_j(2) &= \text{EF}_j(2) \cdot r_j(2) \\ \text{C}_j(3) &= \text{EF}_j(3) \cdot r_j(3) \\ \text{C}'_j(1) &= \text{C}_j(1) \end{aligned}$$

- $\text{ETest}(r, \text{C}, \text{C}', \text{proof}_{\text{Enc}}, \text{proof}_{\text{Class}}, \text{sk}_{\mathcal{T}})$: Given an encoded classification C , if the request is correct, it answers the classification:

$$\text{label} = \bigwedge_j \left((\text{C}'_j(1)^{\text{sk}_{\mathcal{T}}} \neq \text{C}'_j(2)) \wedge (\text{C}'_j(1)^{\text{sk}_{\mathcal{T}}} \neq \text{C}'_j(3)) \right)$$

by making $\text{EQTest}(\text{C}', r(1)^{\text{sk}_{\mathcal{T}}})$ an equality test between the user and the tester. If the request is not correct, it answers a random label.

The request is said correct if $\text{Verif}_{\text{Encode}_{\mathcal{U}}}(\text{param}, r, \text{proof}_{\text{Enc}})$ and $\text{Verif}_{\text{Class}}(\text{EF}, r, \text{C}, \text{C}', \text{proof}_{\text{Class}})$ are correct.

3.2.2 Why does it work?

In this section I will detail how our protocol works and, without enter in the formalism, I will explain why we decided to construct it this way. Last, I will prove its correctness.

If we encrypt two times the database, we can store all the necessary information to construct the formula, namely a first encryption to know if we have a 1 or not and a second to know if we have a 0 or not.

F_j	0		1	
EF_j	$(g^\alpha, h_j, h_0^\alpha)$		$(g^\alpha, h_0^\alpha, h_j)$	
w_j	0	1	0	1
$(r_j(1), r_j(2), r_j(3))$	$(g^\beta, h_0^\beta, h'_j)$	$(g^\beta, h'_j, h_0^\beta)$	$(g^\beta, h_0^\beta, h'_j)$	$(g^\beta, h'_j, h_0^\beta)$
$EF_j(2) \cdot r_j(2)$ power of $EF_j(1) \cdot r_j(1)$	\times	\checkmark	\times	\times
$EF_j(3) \cdot r_j(3)$ power of $EF_j(1) \cdot r_j(1)$	\times	\times	\checkmark	\times
Expected partial classification	\checkmark	\times	\times	\checkmark

F_j	\star	
EF_j	(g^α, h_j, h'_j)	
w_j	0	1
$(r_j(1), r_j(2), r_j(3))$	$(g^\beta, h_0^\beta, h''_j)$	$(g^\beta, h''_j, h_0^\beta)$
$EF_j(2) \cdot r_j(2)$ power of $EF_j(1) \cdot r_j(1)$	\times	\times
$EF_j(3) \cdot r_j(3)$ power of $EF_j(1) \cdot r_j(1)$	\times	\times
Expected partial classification	\checkmark	\checkmark

Table 3.4: Resolution of the first difficulty in section 3.1.3: a bit of a word is positively classified when there is no property between the elements of its encryption.

The Encode_S function allows us to make these two encryptions in a compact manner. The sender will encrypt his future line of the database bit by bit. If the sender wishes to encrypt a bit equal to 1 for example, he could generate two pairs $(g^{\alpha_j}, h_0^{\alpha_j})$ and (g^{β_j}, h_j) . The first pair is used to encrypt a 1 and the second to say that it does not encrypt a 0 but for efficiency, the sender will generate a triple $(g^{\alpha_j}, h_0^{\alpha_j}, h_j)$. The second element of the triple is a power of the first but the third is not. It means that the sender has encrypted a 1. To encrypt a 0, the sender will invert the second and the third elements.

The formal proof is in the next section but we can already feel that the security of the sender's data will lie on the Diffie-Hellman's hypothesis.

To prove that the sender did his job (that he correctly encrypts a series of bits), we make a zero-knowledge proof that shows us that either the second or the third element is h_0 elevating to the power of the same element. That means that a 1 or a 0 was encrypted. Moreover the verifier could see that the sender encrypts a 1 and a 0 at the same time for the same element.

In Prove_S we use the random oracle \mathcal{H} to have a not-interactive zero-knowledge proof. The proof was reduced to be as compact as possible.

As in the section 3.1.3, the learning of the formula consists in the multiplication of the elements in each columns. Because we have triples in each case of the database we multiply according to coordinates.

This was the easiest phase.

To overcome the difficulty to find a protocol that can make learning and classification (cf the first difficulty in 3.1.3), we will classify a word positively when we do not have a property between the elements of the triple as in the table below 3.2.2.

For that, the encryption of a user will be the opposite of the encryption of a sender.

F_j	0		1	
EF_j	$(g^\alpha, h_j, h_0^\alpha)$		$(g^\alpha, h_0^\alpha, h_j)$	
w_j	0	1	0	1
$(r_j(1), r_j(2), r_j(3))$	$(g^\beta, h_0^\beta, h'_j)$	$(g^\beta, h'_j, h_0^\beta)$	$(g^\beta, h_0^\beta, h'_j)$	$(g^\beta, h'_j, h_0^\beta)$
C	$(g^{\alpha\beta}, \cdot, \cdot)$	$(g^{\alpha\beta}, \cdot, h_0^{\alpha\beta})$	$(g^{\alpha\beta}, h_0^{\alpha\beta}, \cdot)$	$(g^{\alpha\beta}, \cdot, \cdot)$
C'	$(g^{\alpha\beta}, \cdot, \cdot)$	$(g^{\alpha\beta}, \cdot, h_0^{\alpha\beta\text{sk}_U})$	$(g^{\alpha\beta}, h_0^{\alpha\beta\text{sk}_U}, \cdot)$	$(g^{\alpha\beta}, \cdot, \cdot)$
$(C'(1)^{\text{sk}_T}, C'(2), C'(3))$	$(g^{\alpha\beta\text{su}}, \cdot, \cdot)$	$(g^{\alpha\beta\text{su}}, \cdot, g^{s\alpha\beta u})$	$(g^{\alpha\beta\text{su}}, g^{s\alpha\beta u}, \cdot)$	$(g^{\alpha\beta\text{su}}, \cdot, \cdot)$
$E\text{Test}_j$	✓	×	×	✓
Expected partial classification	✓	×	×	✓

F_j	★	
EF_j	(g^α, h_j, h'_j)	
w_j	0	1
$(r_j(1), r_j(2), r_j(3))$	$(g^\beta, h_0^\beta, h''_j)$	$(g^\beta, h''_j, h_0^\beta)$
C	$(g^{\alpha\beta}, \cdot, \cdot)$	$(g^{\alpha\beta}, \cdot, \cdot)$
C'	$(g^{\alpha\beta}, \cdot, \cdot)$	$(g^{\alpha\beta}, \cdot, \cdot)$
$E\text{Test}_j$	×	×
Expected partial classification	✓	✓

Table 3.5: Correctness. \cdot symbolise a random element of \mathbb{G}

In Encode_U , the first three coordinates are constructed the same way as in Encode_S but we invert the encryption of a 1 and a 0. The last two coordinates used with the first one correspond to the same encryption but in a different basis. This other basis gives us a trap to form an extractor. That will be very useful, particularly in the security proof that we will see in the next part.

The proof $\text{Prove}_{\text{Encode}_U}$ and the verification $\text{Verif}_{\text{Encode}_U}$ of the right construction of the encryption are formed following the same model as the proof Prove_S and the verification Verif_S , respectively.

As specified in section 3.1.3, the user needs to mix the encryption of his word with the encryption of the formula. This is what makes the **Class** function. C is the product of the two encryptions and C' corresponds to an encryption switch. The tester could answer this last encryption.

$\text{Prove}_{\text{Class}}$ and $\text{Verif}_{\text{Class}}$ are here to ensure the right form of the constructed objects.

The tester will answer a different solution depending on the validity of the proofs $\text{Prove}_{\text{Encode}_U}$ and $\text{Prove}_{\text{Class}}$. That curbs the user.

So to solve the problem of the requested form we use a switching encryption. By doing so, we can give different secrets to the different actors.

To evaluate the correctness, we will show that the tester's answer is the same as the expected classification if the protocol is followed that if the database and the user's encryption are correct. Even if it is not formal, I think it is better to see this property with a table 3.2.2.

3.3 Formally

In this part, I will prove the zero-knowledge proofs, the sender's privacy against a malicious user and the user's privacy against a malicious sender and a honest-but-curious tester.

3.3.1 Zero-knowledge proofs

Proposition 1. *The sender's proof 3.1 of good encoding, user's proof 3.2 of good encoding and user's proof 3.3 of good classification are complete.*

Proof. Easy to check. □

Proposition 2. *The sender's proof 3.1 of good encoding is sound.*

Proof. Let $\text{proof}_S = (c, \{c_{j0}\}_j, \{a_j\}_j, \{a_{j0}\}_j, \{a_{j1}\}_j)$ and $\text{proof}'_S = (c', \{c'_{j0}\}_j, \{a'_j\}_j, \{a'_{j0}\}_j, \{a'_{j1}\}_j)$ be two accepting conversations with $c \neq c'$. For all $j \in \{1, \dots, n\}$:

$$\begin{aligned} A_j &= \begin{cases} g^{a_j} r_j(1)^c & = g^{a_j} g^{\alpha_j c} \\ g^{a'_j} r_j(1)^{c'} & = g^{a'_j} g^{\alpha_j c'} \end{cases} \\ \Rightarrow g^{a_j} g^{\alpha_j c} &= g^{a'_j} g^{\alpha_j c'} \\ \Rightarrow g^{a_j - a'_j} &= g^{\alpha_j (c' - c)} \end{aligned}$$

which gives us a witness $\alpha_j = \frac{a_j - a'_j}{c' - c} \pmod q$ because $c' \neq c$ and q is known. □

Proposition 3. *The sender's proof 3.1 of good encoding verifies the property of honest-verifier zero-knowledgeness.*

Proof. We create a simulator :

- $\text{Sim}_1(\text{param}, r)$: it generates $c \in_R \mathbb{Z}_q$ and for each j , it generates $c_{j0}, a_j, a_{j0}, a_{j1} \in_R \mathbb{Z}_q$, it sets:

$$\begin{aligned} c_{j1} &\leftarrow c - c_{j0} \\ A_j &\leftarrow g^{a_j} r_j(1)^c \\ B_j &\leftarrow h_0^{a_{j0}} r_j(2)^{c_{j0}} \\ C_j &\leftarrow h_0^{a_{j1}} r_j(3)^{c_{j1}} \end{aligned}$$

it defines a random oracle $\mathcal{H}(\{A_j\}_j, \{B_j\}_j, \{C_j\}_j) \leftarrow c$ and returns $(c, \{c_{j0}\}_j, \{a_j\}_j, \{a_{j0}\}_j, \{a_{j1}\}_j)$.

□

Proposition 4. *The user's proof 3.2 of good encoding is sound.*

Proof. Let $\text{proof}_{\text{Enc}} = (c, \{c_{j0}\}_j, \{a_j\}_j, \{a_{j0}\}_j, \{a_{j1}\}_j)$ and $\text{proof}'_{\text{Enc}} = (c', \{c'_{j0}\}_j, \{a'_j\}_j, \{a'_{j0}\}_j, \{a'_{j1}\}_j)$ be two accepting conversations with $c \neq c'$. For all $j \in \{1, \dots, n\}$:

$$\begin{aligned} A_j &= \begin{cases} g^{a_j} r_j(1)^c & = g^{a_j} g^{\alpha_j c} \\ g^{a'_j} r_j(1)^{c'} & = g^{a'_j} g^{\alpha_j c'} \end{cases} \\ \Rightarrow g^{a_j} g^{\alpha_j c} &= g^{a'_j} g^{\alpha_j c'} \\ \Rightarrow g^{a_j - a'_j} &= g^{\alpha_j (c' - c)} \end{aligned}$$

which gives us a witness $\alpha_j = \frac{a_j - a'_j}{c' - c} \bmod q$ because $c' \neq c$ and q is known. \square

Proposition 5. *The user's proof 3.2 of good encoding verifies the property of honest-verifier zero-knowledgeness.*

Proof. We create a simulator :

- $\text{Sim}_2(\text{param}, r)$: it generates $c \in_R \mathbb{Z}_q$ and for each j , it generates $c_{j0}, a_j, a_{j0}, a_{j1} \in_R \mathbb{Z}_q$, it sets:

$$\begin{aligned} c_{j1} &\leftarrow c - c_{j0} \\ A_j &\leftarrow g^{a_j} r_j(1)^c \\ B_j &\leftarrow h_0^{a_{j1}} r_j(2)^{c_{j1}} \\ C_j &\leftarrow h_0^{a_{j0}} r_j(3)^{c_{j0}} \\ D_j &\leftarrow h_1^{a_{j1}} r_j(4)^{c_{j1}} \\ E_j &\leftarrow h_1^{a_{j0}} r_j(5)^{c_{j0}} \end{aligned}$$

and it defines a random oracle $\mathcal{H}(\{A_j\}_j, \{B_j\}_j, \{C_j\}_j, \{D_j\}_j, \{E_j\}_j) \leftarrow c$ and returns $(c, \{c_{j0}\}_j, \{a_j\}_j, \{a_{j0}\}_j, \{a_{j1}\}_j)$. \square

Proposition 6. *The user's proof 3.3 of good classification is sound.*

Proof. Let $\text{proof}_{\text{Class}} = (c, \{a_j\}_j)$ and $\text{proof}'_{\text{Class}} = (c', \{a'_j\}_j)$ be two accepting conversations with $c \neq c'$. For all $j \in \{1, \dots, n\}$:

$$\begin{aligned} A_j &= \begin{cases} C_j(2)^{a_j} C'_j(2)^c & = C_j(2)^{a_j} C_j(2)^{\text{sk}_{\mathcal{U}} c} \\ C_j(2)^{a'_j} C'_j(2)^{c'} & = C_j(2)^{a'_j} C_j(2)^{\text{sk}_{\mathcal{U}} c'} \end{cases} \\ \Rightarrow C_j(2)^{a_j} C_j(2)^{\text{sk}_{\mathcal{U}} c} &= C_j(2)^{a'_j} C_j(2)^{\text{sk}_{\mathcal{U}} c'} \\ \Rightarrow C_j(2)^{a_j - a'_j} &= C_j(2)^{\text{sk}_{\mathcal{U}} (c' - c)} \end{aligned}$$

which gives us a witness $\text{sk}_{\mathcal{U}} = \frac{a_j - a'_j}{c' - c} \bmod q$ because $c' \neq c$ and q is known. \square

Proposition 7. *The user's proof 3.3 of good classification verifies the property of honest-verifier zero-knowledgeness.*

Proof. We create a simulator :

- $\text{Sim}_3(\text{EF}, r, C, C')$: it generates $c \in_R \mathbb{Z}_q$ and for each j , it generates $a_j \in_R \mathbb{Z}_q$, it sets:

$$A_j \leftarrow C_j(2)^{a_j} C'_j(2)^c$$

$$B_j \leftarrow C_j(3)^{a_j} C'_j(3)^c$$

and it defines a random oracle $\mathcal{H}(\{A_j\}_j, \{B_j\}_j) \leftarrow c$ and returns $(c, \{a_j\}_j)$.

□

3.3.2 The security games

Sender's privacy:

The experiment below explains formally how the security of the encoded database works. We considered a malicious user knowing $\text{sk}_{\mathcal{U}}$. The databases is constructed over time and the attacker can ask queries to learn the evaluation of the learned formula on some words.

Let us define the two oracles that define the ideal protocol:

- $\mathcal{O}^{\text{Add}}(w)$: it adds w in DB and outputs that an addition has been successfully done;
- $\mathcal{O}^{\text{Class}}(w)$: it outputs $F(w)$ where F is the learned formula on DB.

The malicious user is not authorised to execute $\text{Encode}_{\mathcal{S}}$ by himself but he is authorised to authenticate and fill DB. He can also perform $\text{Encode}_{\mathcal{U}}$ and Class . The view of the attacker can be simulated by three algorithms: **Initialize**, **Add** and **Test**. **Add** and **Test** queries can be asked as many times as necessary in any order but the **Initialize** query must be called only once at the beginning.

Theorem 8. *Our protocol respect the sender's privacy against a malicious user: there exists a simulator that can, with just access to $\mathcal{O}^{\text{Add}}(w)$ and $\mathcal{O}^{\text{Class}}(w)$, produce an indistinguishable view to the adversary.*

Proof. Step by step, we will construct a simulation of the attacker's view.

Game \mathbf{G}_0 : In the first game, we describe the real situation using our protocol.

- **Initialize**(λ): it executes $\text{param} \leftarrow \text{Setup}(1^\lambda)$: given the security parameter λ , it generates $s \xleftarrow{R} \mathbb{Z}_q$ the master secret key and the parameters $\text{param} = (\mathbb{G}, g, h_0 = g^s, h_1)$ where \mathbb{G} is a group of order q and of generator g and $h_1 \xleftarrow{R} \mathbb{G}$ and generates the user's secret key $\text{sk}_{\mathcal{U}} = u \xleftarrow{R} \mathbb{Z}_q$ and the tester's secret key $\text{sk}_{\mathcal{T}} := s \cdot \text{sk}_{\mathcal{U}}$. It publicizes param , an empty database EDB and an empty learned function EF and outputs $\text{sk}_{\mathcal{U}}$;

- **Add**(w): it computes $(r, \text{proof}_S) \leftarrow \text{Encode}_S(\text{param}, w)$: for each $j \in \{1, \dots, n\}$,

$$r_j := \begin{cases} (g^{\alpha_j}, h_0^{\alpha_j}, h_j) & \text{if } w_j = 1 \\ (g^{\alpha_j}, h_j, h_0^{\alpha_j}) & \text{if } w_j = 0 \end{cases} \quad \text{with } \alpha_j \xleftarrow{R} \mathbb{Z}_q, h_j \xleftarrow{R} \mathbb{G}$$

$$\text{proof}_S \leftarrow \text{Prove}_S(\text{param}, w, r, \{\alpha_j\}_j)$$

It adds (r, proof_S) to EDB, updates $\text{EF} \leftarrow \text{ELearn}(\text{EDB})$: for each $j \in \{1, \dots, n\}$,

$$\text{EF}_j := \left(\prod_{i \in I^+} r_j^{(i)}(1), \prod_{i \in I^+} r_j^{(i)}(2), \prod_{i \in I^+} r_j^{(i)}(3) \right)$$

and outputs (r, proof_S) ;

- **Test**($r, \text{proof}_{\text{Enc}}, C, C', \text{proof}_{\text{Class}}$): it computes $\text{label} \leftarrow \text{ETest}(r, \text{proof}_{\text{Enc}}, C, C', \text{proof}_{\text{Class}}, \text{EF}, \text{sk}_T)$: if the $\text{proof}_{\text{Enc}}$ and $\text{proof}_{\text{Class}}$ are correct, it executes the tester's part in the equality test $\text{EQTest}(C', r(1)^{\text{sk}_T})$ and answers:

$$\text{label} = \bigwedge_j \left((C'_j(1)^{\text{sk}_T} \neq C'_j(2)) \wedge (C'_j(1)^{\text{sk}_T} \neq C'_j(3)) \right)$$

If the request is not correct, it answers a random label.

Game G_1 : In this game, we modify in **Add**() the simulator by using $\mathcal{O}^{\text{Add}}()$.

- **Add**(w): it performs $\mathcal{O}^{\text{Add}}(w)$, computes $(r, \text{proof}_S) \leftarrow \text{Encode}_S(\text{param}, w)$ and outputs (r, proof_S) ;

This game is perfectly indistinguishable from the previous one since only formal modification.

$$\text{Adv}_{G_1}^{\text{IND}_{\text{sender}}}(\mathcal{A}) = \text{Adv}_{G_0}^{\text{IND}_{\text{sender}}}(\mathcal{A})$$

Game G_2 : In this game, we modify in **Initialize** how **Setup**(1^λ) works. That allows us to use $\mathcal{O}^{\text{Class}}$ in **Test**.

- **Initialize**(λ): given a security parameter λ , it generates $s \xleftarrow{R} \mathbb{Z}_q$ the master secret key and the parameters $\text{param} = (\mathbb{G}, g, h_0 = g^s, h_1 = g^k)$ where \mathbb{G} is a group of order q and of generator g and $k \xleftarrow{R} \mathbb{Z}_q$. It generates the user's secret key $\text{sk}_U = u \xleftarrow{R} \mathbb{Z}_q$ and the tester's secret key $\text{sk}_T := s \cdot \text{sk}_U$. It publicizes param , an empty database EDB and an empty learned function EF and outputs sk_U ;
- **Test**($r, \text{proof}_{\text{Enc}}, C, C', \text{proof}_{\text{Class}}$): if $\text{proof}_{\text{Enc}}$ and $\text{proof}_{\text{Class}}$ are correct, by using the trapdoor k , it can decode r . For all $j \in \{1..n\}$, it defines:

$$w_j := \begin{cases} 0 & \text{if } r_j(1)^k = r_j(4) \\ 1 & \text{if } r_j(1)^k = r_j(5) \end{cases}$$

During the interactions with the attacker, it outputs random elements. Then, it outputs $\text{label} \leftarrow \mathcal{O}^{\text{Class}}(w)$. If $\text{proof}_{\text{Enc}}$ and $\text{proof}_{\text{Class}}$ are not correct, it outputs a random value label .

The statistical soundness of $\text{proof}_{\text{Enc}}$ and $\text{proof}_{\text{Class}}$ ensures that:

$\forall j \in \{1, \dots, n\}, \exists \alpha_j :$

$$(r_j(1) = g^{\alpha_j}) \wedge \left[((r_j(2) = h_0^{\alpha_j}) \wedge (r_j(4) = h_1^{\alpha_j})) \vee ((r_j(3) = h_0^{\alpha_j}) \wedge (r_j(5) = h_1^{\alpha_j})) \right] \\ \wedge (r_j(2) \neq r_j(3)) \wedge (r_j(4) \neq r_j(5))$$

and

$\forall j \in \{1, \dots, n\}, \exists \text{sk}_{\mathcal{U}} :$

$$(C_j(1) = \text{EF}_j(1) \cdot r_j(1)) \wedge (C_j(2) = \text{EF}_j(2) \cdot r_j(2)) \wedge (C_j(3) = \text{EF}_j(3) \cdot r_j(3)) \\ \wedge (C'_j(1) = C_j(1)) \wedge (C'_j(2) = C_j(2)^{\text{sk}_{\mathcal{U}}}) \wedge (C'_j(3) = C_j(3)^{\text{sk}_{\mathcal{U}}})$$

So, this game is statistically indistinguishable from the previous one.

$$\text{Adv}_{\mathbf{G}_2}^{\text{IND}_{\text{sender}}}(\mathcal{A}) = \text{Adv}_{\mathbf{G}_1}^{\text{IND}_{\text{sender}}}(\mathcal{A})$$

Game \mathbf{G}_3 : In **Add**, we modify the proof used in $\text{Encode}_{\mathcal{S}}$ by its simulation 3.3.1.

- **Add**(w): it performs $\mathcal{O}^{\text{Add}}(w)$, computes $(r, \text{proof}_{\mathcal{S}})$ with $\text{proof}_{\mathcal{S}} = \text{Sim}_1(\text{param}, r)$ and outputs $(r, \text{proof}_{\mathcal{S}})$;

$$\left| \text{Adv}_{\mathbf{G}_3}^{\text{IND}_{\text{sender}}}(\mathcal{A}) - \text{Adv}_{\mathbf{G}_2}^{\text{IND}_{\text{sender}}}(\mathcal{A}) \right| \leq \frac{(q_{\mathcal{H}} + m)m}{q}$$

where m is the number of **Add**-queries and $q_{\mathcal{H}}$ the number of oracle queries used in Sim_1 .

Game \mathbf{G}_4 : In this game, we modify in **Add**(w) how $\text{Encode}_{\mathcal{S}}$ works.

- **Add**(w): it performs $\mathcal{O}^{\text{Add}}(w)$, computes $(r, \text{proof}_{\mathcal{S}})$ with for each $j \in \{1, \dots, n\}$:

$$r_j := (A_j, B_j, C_j) \text{ with } A_j, B_j, C_j \xleftarrow{R} \mathbb{G}$$

and outputs $(r, \text{proof}_{\mathcal{S}})$;

We use the lemma:

Lemma 1.

$$\left| \text{Adv}_{\mathbf{G}_4}^{\text{IND}_{\text{sender}}}(\mathcal{A}) - \text{Adv}_{\mathbf{G}_3}^{\text{IND}_{\text{sender}}}(\mathcal{A}) \right| \leq \text{Adv}^{\text{DDH}}(\mathcal{D}) \leq \text{Adv}^{\text{DDH}}(t_{\mathcal{A}} + [10nm + nm'] \cdot t_{\text{exp}})$$

where m' is the number of **Test**-queries.

Proof of lemma. **Game G_0 :** We consider this situation again:

- **Add(w):** it performs $\mathcal{O}^{\text{Add}}(w)$, computes (r, proof_S) by: for each $j \in \{1, \dots, n\}$,

$$r_j := \begin{cases} (g^{\alpha_j}, h_0^{\alpha_j}, h_j) & \text{if } w_j = 1 \\ (g^{\alpha_j}, h_j, h_0^{\alpha_j}) & \text{if } w_j = 0 \end{cases} \quad \text{with } \alpha_j \xleftarrow{R} \mathbb{Z}_q, h_j \xleftarrow{R} \mathbb{G}$$

$$\text{proof}_S \leftarrow \text{Sim}_1(\text{param}, r)$$

and outputs (r, proof_S) ;

Game G_1 : In this game, we modify in **Add(w)** how **Encode $_S$** works.

- **Add(w):** it performs $\mathcal{O}^{\text{Add}}(w)$, computes (r, proof_S) by for each $j \in \{1, \dots, n\}$:

$$r_j := \begin{cases} (g^{\gamma_j} g^{\alpha_j}, h_0^{\gamma_j} h_0^{\alpha_j}, h_j) & \text{if } w_j = 1 \\ (g^{\gamma_j} g^{\alpha_j}, h_j, h_0^{\gamma_j} h_0^{\alpha_j}) & \text{if } w_j = 0 \end{cases} \quad \text{with } \alpha_j, \gamma_j \xleftarrow{R} \mathbb{Z}_q, h_j \xleftarrow{R} \mathbb{G}$$

and outputs (r, proof_S) ;

Because $\alpha_j \leftarrow \alpha_j - \gamma_j$ is a bijective substitution:

$$\text{Adv}_{G_1}^{\text{IND}_{\text{sender}}}(\mathcal{A}) = \text{Adv}_{G_0}^{\text{IND}_{\text{sender}}}(\mathcal{A})$$

Game G_2 : In this game, the simulator takes in input (A, B, C, D) that is a DDH tuple and we modify the **Initialize**-query and the **Add**-query.

- **Initialize(λ):** Given a security parameter λ , it generates $s \xleftarrow{R} \mathbb{Z}_q$ the master secret key and the parameters $\text{param} = (\mathbb{G}, A, B, A^k)$ where \mathbb{G} is a group of order q and $k \xleftarrow{R} \mathbb{Z}_q$. It generates the user's secret key $\text{sk}_U = u \xleftarrow{R} \mathbb{Z}_q$ and the tester's secret key $\text{sk}_T := s \cdot \text{sk}_U$. It publicizes param , an empty database EDB and an empty learned function **EF** and outputs sk_U ;
- **Add(w):** it performs $\mathcal{O}^{\text{Add}}(w)$, computes (r, proof_S) by for each $j \in \{1, \dots, n\}$:

$$r_j := \begin{cases} (A^{\gamma_j} C^{\alpha_j}, B^{\gamma_j} D^{\alpha_j}, h_j) & \text{if } w_j = 1 \\ (A^{\gamma_j} C^{\alpha_j}, h_j, B^{\gamma_j} D^{\alpha_j}) & \text{if } w_j = 0 \end{cases} \quad \text{with } \alpha_j, \gamma_j \xleftarrow{R} \mathbb{Z}_q, h_j \xleftarrow{R} \mathbb{G}$$

$$\text{proof}_S := \text{Sim}_1(\text{param}, r_j)$$

and outputs (r, proof_S) ;

When (A, B, C, D) is a Diffie-Hellman tuple, this game is perfectly indistinguishable from the previous one because $\alpha_j \leftarrow \frac{\alpha_j}{\log_A(C)}$ is a bijective substitution. So we have:

$$\text{Adv}_{G_2}^{\text{IND}_{\text{sender}}}(\mathcal{A}) = \text{Adv}_{G_1}^{\text{IND}_{\text{sender}}}(\mathcal{A})$$

or $\det M = (xy' - yx)^{mn} = x^{mn}(y' - y)^{mn} \neq 0$ because $y' \neq y$ by assumption. Then M is invertible.

Game G_4 : In this game, we modify the encoding of the database:

- **Add(w):** it performs $\mathcal{O}^{\text{Add}}(w)$, computes (r, proof_S) by for each $j \in \{1, \dots, n\}$:

$$r_j := (A^{a_j}, A^{b_j}, A^{c_j}) \text{ with } a_j, b_j, c_j \xleftarrow{R} \mathbb{Z}_q$$

and outputs (r, proof_S) ;

This game is identical from the previous one because there exists a permutation P such that:

$$\begin{pmatrix} a_{11} \\ a_{12} \\ \dots \\ a_{mn} \\ b_{11} \\ b_{12} \\ \dots \\ b_{mn} \\ c_{11} \\ c_{12} \\ \dots \\ c_{mn} \end{pmatrix} = PM \times \begin{pmatrix} \gamma_{11} \\ \alpha_{11} \\ \gamma_{12} \\ \alpha_{12} \\ \dots \\ \gamma_{mn} \\ \alpha_{mn} \\ z_{11} \\ z_{12} \\ \dots \\ z_{mn} \end{pmatrix}$$

and because PM is invertible.

$$0 = \text{Adv}_{G_4}^{\text{IND}_{\text{sender}}}(\mathcal{A}) = \text{Adv}_{G_3}^{\text{IND}_{\text{sender}}}(\mathcal{A})$$

So,

$$\text{Adv}_{G_0}^{\text{IND}_{\text{sender}}}(\mathcal{A}) \leq \text{Adv}^{\text{DDH}}(t_{\mathcal{A}} + [10nm + nm'] \cdot t_{\text{exp}})$$

□

At the end, the simulated view of the attacker is:

- **Add(w):** it performs $\mathcal{O}^{\text{Add}}(w)$, computes (r, proof_S) with for each $j \in \{1, \dots, n\}$:

$$\begin{aligned} r_j &:= (A_j, B_j, C_j) \text{ with } A_j, B_j, C_j \xleftarrow{R} \mathbb{G} \\ \text{proof}_S &\leftarrow \text{Sim}_1(\text{param}, r) \end{aligned}$$

and outputs (r, proof_S) ;

- **Test**($r, \text{proof}_{\text{Enc}}, C, C', \text{proof}_{\text{Class}}$): if $\text{proof}_{\text{Enc}}$ and $\text{proof}_{\text{Class}}$ are correct, by using the trapdoor k , it can decode r . For all $j \in \{1..n\}$, it defines:

$$w_j := \begin{cases} 0 & \text{if } r_j(1)^k = r_j(4) \\ 1 & \text{if } r_j(1)^k = r_j(5) \end{cases}$$

During the interactions with the attacker, it outputs random elements. Then, it outputs $\text{label} \leftarrow \mathcal{O}^{\text{Class}}(w)$. If $\text{proof}_{\text{Enc}}$ and $\text{proof}_{\text{Class}}$ are not correct, it outputs a random value label .

This game uses \mathcal{O}^{Add} and $\mathcal{O}^{\text{Class}}$ and because all the elements of the database are taken independent and random, we have randomly simulated the view of the attacker.

$$\text{Adv}_{\mathbf{G}_0}^{\text{IND}_{\text{sender}}}(\mathcal{A}) \leq \text{Adv}^{\text{DDH}}(t_{\mathcal{A}} + [10nm + nm'] \cdot t_{\text{exp}}) + \frac{(q_{\mathcal{H}} + m)m}{q}$$

□

3.4 User's privacy:

The experiment below explains formally how the security of the user's request works. We consider an attacker that can play the role of a malicious sender. The database is constructed over time.

We recall the two oracles that define the ideal protocol:

- $\mathcal{O}^{\text{Add}}(w)$: it adds w in DB and outputs that an addition has been successfully done;
- $\mathcal{O}^{\text{Class}}(w)$: it outputs $F(w)$ where F is the learned formula on DB.

The attacker is authorised to execute $\text{Encode}_{\mathcal{S}}$ by himself. He can also perform $\text{Encode}_{\mathcal{U}}$ but he can not execute the algorithm $\text{Class}(\text{EF}, r, \text{sk}_{\mathcal{U}})$ of our protocol because he does not know $\text{sk}_{\mathcal{U}}$. The view of the attacker can be simulated by three algorithms: $\text{Initialize}(\lambda)$, $\text{Add}(r, \text{proof}_{\mathcal{S}})$ and $\text{Class}(w)$. Add and Class queries can be asked as many times as necessary in any order but the Initialize query must be called only once at the beginning.

Theorem 9. *Our protocol respect the user's privacy against a malicious sender: there exists a simulator that can, with just access to $\mathcal{O}^{\text{Add}}(w)$ and $\mathcal{O}^{\text{Class}}(w)$, produce an indistinguishable view to the adversary.*

Proof. Step by step, we will construct a simulation of the attacker's view.

Game \mathbf{G}_0 : In the first game, we describe the real situation using our protocol.

- **Initialize**(λ): it executes $\text{param} \leftarrow \text{Setup}(1^\lambda)$: given the security parameter λ , it generates $s \xleftarrow{R} \mathbb{Z}_q$ the master secret key and the parameters $\text{param} = (\mathbb{G}, g, h_0 = g^s, h_1)$ where \mathbb{G} is a group of order q and of generator g and $h_1 \xleftarrow{R} \mathbb{G}$ and generates the user's secret key $u \xleftarrow{R} \mathbb{Z}_q$ and the tester's secret key $\text{sk}_T := s \cdot \text{sk}_U$. It publicizes param , an empty database EDB and an empty learned function EF;
- **Add**(r, proof_S): it adds (r, proof_S) to EDB, updates $\text{EF} \leftarrow \text{ELearn}(\text{EDB})$: for each $j \in \{1, \dots, n\}$,

$$\text{EF}_j := \left(\prod_{i \in I^+} r_j^{(i)}(1), \prod_{i \in I^+} r_j^{(i)}(2), \prod_{i \in I^+} r_j^{(i)}(3) \right)$$

and outputs that an addition has been successfully done;

- **Class**(w): it computes $(r, \text{proof}_{\text{Enc}}) \leftarrow \text{Encode}_U(\text{param}, w)$: for each $j \in \{1, \dots, n\}$,

$$r_j := \begin{cases} (g^{\alpha_j}, h_0^{\alpha_j}, h_j, h_1^{\alpha_j}, h'_j) & \text{if } w_j = 0 \\ (g^{\alpha_j}, h_j, h_0^{\alpha_j}, h'_j, h_1^{\alpha_j}) & \text{if } w_j = 1 \end{cases} \quad \text{with } \alpha_j \xleftarrow{R} \mathbb{Z}_q, h_j, h'_j \xleftarrow{R} \mathbb{G}$$

$$\text{proof}_{\text{Enc}} := \text{Prove}_{\text{Encode}_U}(\text{param}, w, r, \{\alpha_j\}_j)$$

and $(C, C', \text{proof}_{\text{Class}}) \leftarrow \text{Class}(\text{EF}, r, \text{sk}_U)$: for each $j \in \{1, \dots, n\}$,

$$C_j := (\text{EF}_j(1) \cdot r_j(1), \text{EF}_j(2) \cdot r_j(2), \text{EF}_j(3) \cdot r_j(3))$$

$$C'_j := (C_j(1), C_j(2)^{\text{sk}_U}, C_j(3)^{\text{sk}_U})$$

$$\text{proof}_{\text{Class}} := \text{Prove}_{\text{Class}}(\text{EF}, r, C, C', \text{sk}_U)$$

It outputs $(r, \text{proof}_{\text{Enc}}, C, C', \text{proof}_{\text{Class}}, \text{label})$.

Game G₁: In this game, we modify in **Add**() the simulator by using $\mathcal{O}^{\text{Add}}()$.

- **Add**(w): it performs $\mathcal{O}^{\text{Add}}(w)$ and outputs that an addition has been successfully done;

This game is perfectly indistinguishable from the previous one since only formal modification.

$$\text{Adv}_{\mathbf{G}_1}^{\text{IND}_{\text{user}}}(\mathcal{A}) = \text{Adv}_{\mathbf{G}_0}^{\text{IND}_{\text{user}}}(\mathcal{A})$$

Game G₂: In **Class**, we modify the proof used in **Encode_U** and **Class** by their simulation 3.3.1 3.3.1.

- **Class**(w): it computes $(r, \text{proof}_{\text{Enc}}) \leftarrow \text{Encode}_U(\text{param}, w)$ with $\text{proof}_{\text{Enc}} \leftarrow \text{Sim}_2(\text{param}, r)$ and $(C, C', \text{proof}_{\text{Class}}) \leftarrow \text{Class}(\text{EF}, r, \text{sk}_U)$ with $\text{proof}_{\text{Class}} \leftarrow \text{Sim}_3(\text{EF}, r, C, C')$. It outputs $(r, \text{proof}_{\text{Enc}}, C, C', \text{proof}_{\text{Class}}, \text{label})$.

$$|\text{Adv}_{\mathbf{G}_2}^{\text{IND}_{\text{user}}}(\mathcal{A}) - \text{Adv}_{\mathbf{G}_1}^{\text{IND}_{\text{user}}}(\mathcal{A})| \leq \frac{(q_{2\mathcal{H}} + m)m}{q} + \frac{(q_{3\mathcal{H}} + m)m}{q}$$

where m is the number of **Class**-queries, $q_{2\mathcal{H}}$ the number of oracle queries used in Sim_2 and $q_{3\mathcal{H}}$ the number of oracle queries used in Sim_3 .

Game \mathbf{G}_3 : In this game, we modify **Class**(w).

- **Class**(w): it randomly takes r, \mathbf{C} , it computes $\text{label} \leftarrow \mathcal{O}^{\text{Class}}(w)$ and \mathbf{C}' by:

$$\mathbf{C}'_j := \begin{cases} (A_j, B_j, C_j) & \text{if } \text{label} = 1 \\ (A_j, A_j^{\text{sk}_\tau}, B_j) & \text{if } \text{label} = 0 \end{cases} \quad \text{with } , A_j, B_j, C_j \xleftarrow{R} \mathbb{G}$$

it computes $\text{proof}_{\text{Enc}} \leftarrow \text{Sim}_2(\text{param}, r)$ and $\text{proof}_{\text{Class}} \leftarrow \text{Sim}_3(EF, r, C, \mathbf{C}')$ and outputs $(r, \text{proof}_{\text{Enc}}, \mathbf{C}, \mathbf{C}', \text{proof}_{\text{Class}}, \text{label})$.

By the Lemma 1, we have that:

$$|\text{Adv}_{\mathbf{G}_3}^{\text{IND}_{\text{user}}}(\mathcal{A}) - \text{Adv}_{\mathbf{G}_2}^{\text{IND}_{\text{user}}}(\mathcal{A})| \leq \text{Adv}^{\text{DDH}}(t_{\mathcal{A}} + 18nm \cdot t_{\text{exp}})$$

At the end, the simulated view of the attacker is:

- **Add**(w): it performs $\mathcal{O}^{\text{Add}}(w)$ and outputs that an addition has been successfully done;
- **Class**(w): **Class**(w): it randomly takes r, \mathbf{C} , it computes $\text{label} \leftarrow \mathcal{O}^{\text{Class}}(w)$ and \mathbf{C}' by:

$$\mathbf{C}'_j := \begin{cases} (A_j, B_j, C_j) & \text{if } \text{label} = 1 \\ (A_j, A_j^{\text{sk}_\tau}, B_j) & \text{if } \text{label} = 0 \end{cases} \quad \text{with } , A_j, B_j, C_j \xleftarrow{R} \mathbb{G}$$

it computes $\text{proof}_{\text{Enc}} \leftarrow \text{Sim}_2(\text{param}, r)$ and $\text{proof}_{\text{Class}} \leftarrow \text{Sim}_3(EF, r, C, \mathbf{C}')$ and outputs $(r, \text{proof}_{\text{Enc}}, \mathbf{C}, \mathbf{C}', \text{proof}_{\text{Class}}, \text{label})$.

This game uses \mathcal{O}^{Add} and $\mathcal{O}^{\text{Class}}(w)$ and because all the elements in output of the **Class**-queries are taken independent and random, we have randomly simulated the view of the attacker.

$$\text{Adv}_{\mathbf{G}_0}^{\text{IND}_{\text{user}}}(\mathcal{A}) \leq \text{Adv}^{\text{DDH}}(t_{\mathcal{A}} + 18nm \cdot t_{\text{exp}}) + \frac{(q_{1\mathcal{H}} + q_{2\mathcal{H}} + 2m)m}{q}$$

□

Conclusion and future work

This conclusion is in two parts. In the first one, I will take a step back and talk about the work we did on machine learning, and more precisely on the consistency model. In the second par, I will share my personal conclusion about my internship.

4.1 General conclusion

The emergence of the home automation invites leaning algorithms in our houses. They are at the core of data manipulation in the health or bank field. To protect privacy, research is necessary, as we made, on the uses of such algorithms combined with cryptology. Obtaining efficient protocols is not an easy task. Our protocol does not seem practical to our model. As if the fact that the model is already weak because it does not accept error was not enough, this learning model makes memorisation but does not provide generalisation. Moreover our protocol requires a kind of double encryptions and the need to prove all the constructions increases the amount of traffic (cf fig. 4.1). However, our work has a lot of advantages. The first one which was a given criterion, is that all of our solution does not use FHE. Then, we have the learning on encrypted data, the protection of the learned formula (in our model this was essential to obtain a solution) against malicious adversaries and the classification on encrypted data. The primary difficulty was to find a protocol that curb the user without giving too much power to the tester and my main difficulty was to prove the security of our protocol, which was something new to me..

Size of the send data ($n = \text{nb of bits}$)	nb of elt of \mathbb{G}	nb of elt of \mathbb{Z}_q
$\mathcal{S} \rightarrow \text{EDB} : \text{per line}$	$3n$	$4n + 1$
$\mathcal{U} \rightarrow \mathcal{T} : \text{per request}$	$11n$	$7n$

4.2 Personal conclusion

By means of this internship, I discovered a new field in which cryptography can be used. I have been able to learn both with online courses and books but also through the project we had to achieve at the same time. I had often heard the expression “machine learning” but without knowing what it really meant. Now I can rudely talk about it, and I am glad I have an open door to this vast field offering a wide variety of opportunities in cryptology.

Thanks to the team, I acquired knowledge in many areas, because the open space allows numerous exchanges. Furthermore, the preparation of the Eurocrypt international conference promoted team building. It was also the occasion for me to attend presentations made by members of the team or by guests and to attend workshops, which are good opportunities to discuss and to enjoy kinds of small private courses.

Regarding cryptology, the objectives of my internship are, I think, achieved. By being able to go from building the protocol to writing the security proof, I am now aware of what the research work really is, and how different it is from the university where we attend presentations of already established results. I also learned a lot during the writing of the paper. I had to understand the mechanism of a game proof, and then write it. David guided me through the steps, and corrected the proofs several times by providing me advice. I could say that this internship was complementary to the lessons I had during my master's degree. Thanks to my courses, I was able to feel that our protocol was going to work, but I had to make the formal proof of it, and it is at the ENS that I learned how to do it.

About the research, it is very gratifying to find a result. In my case it is a protocol based on a simple model but during those months of searching and questioning I managed to see the extend and the subtleties of the issues I had to overcome. Many of which don't depend upon the model we solved, but on the contrary come from the general domain we want to solve. However, all this work and those new working methods I acquired will be useful when I will start my thesis on the following subject: Big data and privacy.

I am very glad of the fulfillment of the project and of the objectives of the internship. Our work will be continued during the next three years, especially with avenues of research about classification with neural networks and encrypted data. Finally, I would like to reiterate my sincere thanks to my supervisors for their support and their advice.

Bibliography

- [FS87] Amos Fiat and Adi Shamir. *How To Prove Yourself: Practical Solutions to Identification and Signature Problems*, pages 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987.
- [GMR85] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.
- [Gol06] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, New York, NY, USA, 2006.
- [Knu90] Donald Knuth. *TUGboat*, volume 11, pages 497–498. TUGboat, 1990.
- [Riv93] Ronald L. Rivest. *Cryptography and machine learning*, pages 427–439. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.
- [Sch14] Rob Schapire. Computer science 511 - theoretical machine learning, 2014.
- [SQRS17] Jure Sokolic, Qiang Qiu, M.R.D. Rodrigues, and Guillermo Sapiro. Learning to succeed while teaching to fail: Privacy in closed machine learning systems. 05 2017.

Bonus: works on Matlab

In this section, I will present the work I did with Matlab to understand and manipulate different kinds of learning algorithms. There is no cryptography here but the knowledge I acquired through this work is part of the preparation to my PhD. I made 4 TPs in supervised machine learning corresponding to the 4 sub-parts below.

5.1 Linear regression

The first exercise was to implement linear regression with one variable. The goal was to predict the profits of a food truck from data containing the profits and the populations of cities.

Regression is to specify that the learned function needs to be a continuous function and linear is because in this case, we wanted to learn a linear function.

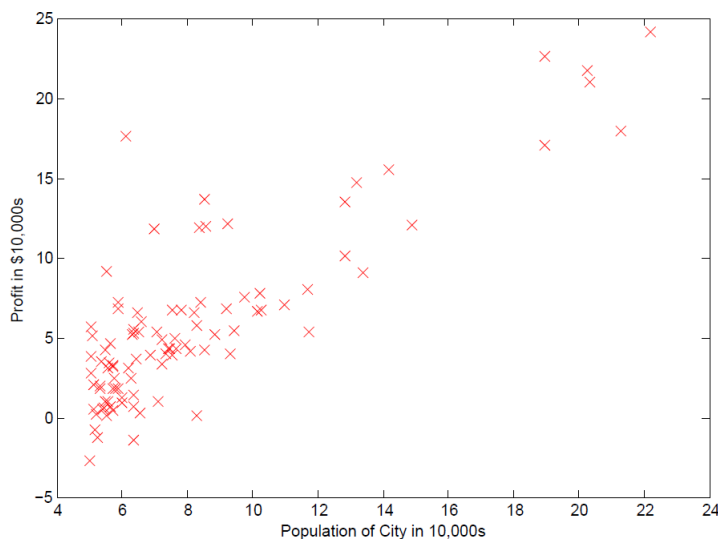


Figure 5.1: First learning problem

From the data set $\{(x^{(i)}, y^{(i)}) | i \in [1..m]\}$ we wanted to find a linear function that approximates the labels $y^{(i)}$ on the input $x^{(i)}$.

Let us consider the hypothesis h_θ :

$$h_\theta(x) = \sum_{j=1}^n \theta_j x_j$$

We want to adjust the parameters $\theta = (\theta_0, \dots, \theta_n)$ and once the best ones are found, h_θ will become the learned function. Because we want to minimise the distance between $x^{(i)}$ and $y^{(i)}$, we can define the cost function:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

and our objective is to find the minimum of J .

The method used in this TP was to compute the minimum of the function J by the gradient descent: update θ by for each $j \in \{1, \dots, n\}$:

$$\begin{aligned} \theta_j &:= \theta_j - \alpha \frac{1}{m} \frac{\partial J(\theta)}{\partial \theta_j} \\ \Rightarrow \begin{cases} \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \\ \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \end{cases} \end{aligned}$$

After having made enough iterations, the θ found will be close enough to the minimal θ . This method works because J is convex. The convergence speed is determined by the parameter α . This is an arbitrary interval to be carefully determined: if it is too large, it is possible to skip the minimum and to diverge, and if it is too short, the algorithm will take a lot of time which is not needed if the length of the interval is well suited in the first place.

```

1  % Part of TP1, the data are in a vectorized form
2
3  data = load('ex1data1.txt'); % the labelled data are given in a file
4  X = data(:, 1);
5  y = data(:, 2);
6  m = length(y); % number of training examples
7
8  X = [ones(m, 1), data(:,1)]; % to consider theta_0
9  theta = zeros(2, 1); % parameters that we want to find
10 iterations = 1500;
11 alpha = 0.01;
12
13 function J = computeCostMulti(X, y, theta)
14     m = length(y);
15     J = 1/(2*m) * (X*theta-y) .'*(X*theta-y);
16 end
17
18 J = computeCost(X, y, theta);
19
20 function [theta, J_history] = gradientDescent(X, y, theta, alpha,
21     num_iters)
22     m = length(y);

```

```

22     J_history = zeros(num_iters, 1);
23
24     for iter = 1:num_iters
25         temp0 = theta(1) - alpha*1/m*sum((X*theta-y).'*X(:,1));
26         temp1 = theta(2) - alpha*1/m*sum((X*theta-y).'*X(:,2));
27         theta = [temp0; temp1];
28         J_history(iter) = computeCost(X, y, theta);
29     end
30 end
31
32 theta = gradientDescent(X, y, theta, alpha, iterations);
33
34 % to plot the training data and linear regression
35 hold on;
36 plot(X(:,2), X*theta, '-')
37 legend('Training data', 'Linear regression')
38 hold off
39
40 % the prediction values for population sizes of 35,000 and 70,000 are
    obtained with
41 predict1 = [1, 3.5] *theta;
42 predict2 = [1, 7] * theta;

```

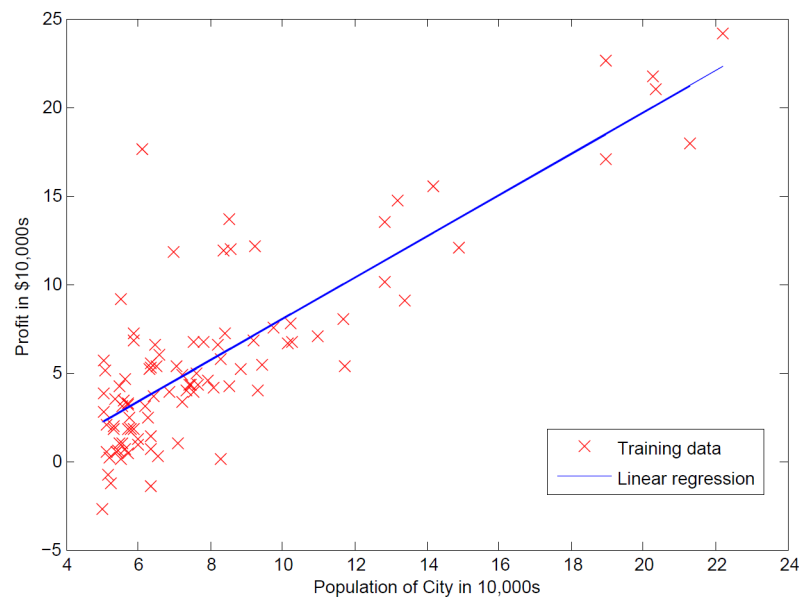


Figure 5.2: First learning problem with linear learned function

In the TP1, I also made the same exercise in multiple variables.

5.2 Logistic regression

The second exercise was to implement a logistic regression model. The goal was to obtain a probability to help a university to decide if a new student is accepted or rejected from the results of two exams. The dataset is the set of applicant's scores on two exams with its admission decision.

In the first TP, the learning model mapped to \mathbb{R} . To solve this second situation we needed another kind of model that maps to $\{0, 1\}$ to have probabilities.

The logistic function is a function used in a lot of fields and in particular in statistics because it represents the probability for the output to be 1.

Logistic function:

$$g(z) = \frac{1}{1 + e^{-z}}$$

We defined the hypothesis function with the logistic function by:

$$h_{\theta}(x) = g(\theta^T X)$$

Like that, $0 < h_{\theta}(x) < 1$ and h_{θ} is better suited for classification. By defining a threshold, we can decide if an applicant is accepted or rejected.

With its new hypothesis function, we needed to define J again:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m [(-y^{(i)} \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})))]$$

This time the partial derivatives for the gradient descent are:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

and the gradient descent is always the repetition of:

$$\{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)\}$$

```

1 % Part of TP2
2
3 data = load('ex2data1.txt');
4 X = data(:, [1, 2]);
5 y = data(:, 3);
6 [m, n] = size(X);
7 X = [ones(m, 1) X];
8
9 initial_theta = [-24; 0.2; 0.2]; % non-zero theta
10
11 function g = sigmoid(z)
12     g = 1./(1+exp(-z));

```

```
13 end
14
15 function [J, grad] = costFunction(theta, X, y)
16     m = length(y); % number of training examples
17     J = 0;
18     grad = zeros(size(theta));
19
20     J = 1/m * sum(-y.*log(sigmoid(X*theta)) - (1-y).*log(1-sigmoid(
21         X*theta)));
22
23     for i = 1:size(X,2)
24         grad(i) = 1/m * sum((sigmoid(X*theta)-y).'*X(:,i));
25     end
26
27 [cost, grad] = costFunction(initial_theta, X, y);
28
29 options = optimset('GradObj', 'on', 'MaxIter', 400);
30 [theta, cost] = fminunc(@(t)(costFunction(t, X, y)), initial_theta,
31     options);
32
33 % the prediction probability for a student with score 45 on exam 1
34 % and score 85 on exam 2
35
36 prob = sigmoid([1 45 85] * theta);
37
38 % to compute accuracy on the training set
39 function p = predict(theta, X)
40
41     m = size(X, 1); % Number of training examples
42     p = zeros(m, 1);
43
44     for i = 1:m
45         if sigmoid(X(i,:)*theta) >= 0.5
46             p(i) = 1;
47         else
48             p(i) = 0;
49         end
50     end
51
52 p = predict(theta, X);
```

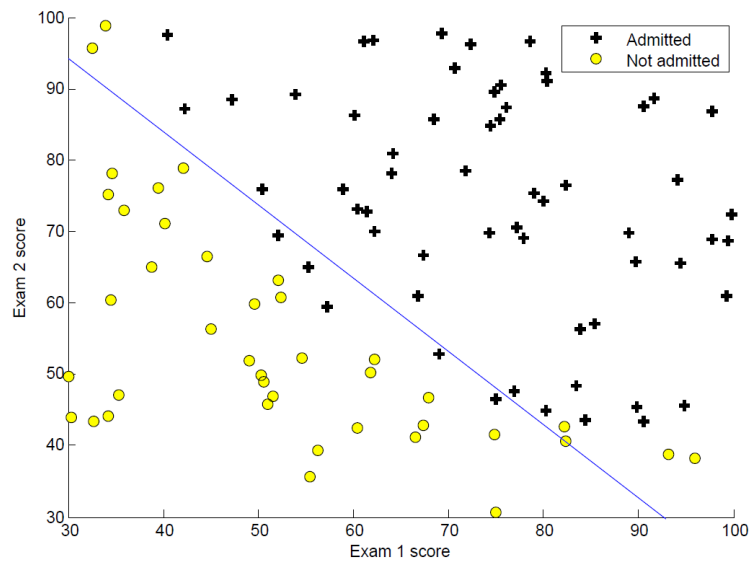


Figure 5.3: Second linear learning problem

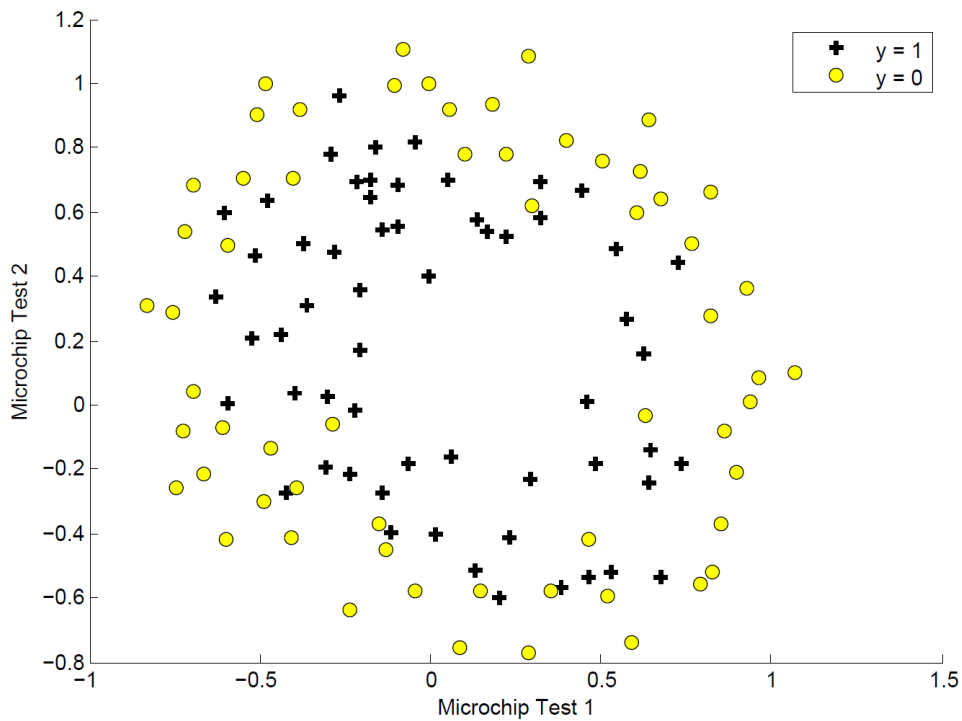


Figure 5.4: Non-linear learning problem

In the second exercise, linear regression was impossible because of the structure of the data 5.2. So we used polynomial terms until the sixth power with the `mapFeature()` function. This forced us to improve the efficiency of the learning algorithm. To take

advantage of the power of the implementation of Matlab, we used the `fminunc()` function to compute the gradient descent.

The regularized cost function is:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m [(-y^{(i)} \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})))] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

And in this case the partial derivatives are:

$$\begin{cases} \frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \\ \frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \end{cases}$$

```

1 % Part of TP2
2
3 data = load('ex2data2.txt');
4 X = data(:, [1, 2]);
5 y = data(:, 3);
6 [m, n] = size(X);
7 X = [ones(m, 1) X];
8
9 X = mapFeature(X(:,1), X(:,2));
10 initial_theta = zeros(size(X, 2), 1);
11 lambda = 1;
12
13 function [J, grad] = costFunctionReg(theta, X, y, lambda)
14     m = length(y); % number of training examples
15     J = 0;
16     grad = zeros(size(theta));
17
18     J = 1/m * sum(-y.*log(sigmoid(X*theta)) - (1-y).'*log(1-sigmoid(
19         X*theta))) + lambda/(2*m)*(sum(theta.^2) - theta(1)^2);
20     grad(1) = 1/m * sum((sigmoid(X*theta) - y).'*X(:,1));
21
22     for i = 2:size(X,2)
23         grad(i) = 1/m * sum((sigmoid(X*theta) - y).'*X(:,i)) + lambda/m *
24             theta(i);
25     end
26
27 end
28
29 options = optimset('GradObj', 'on', 'MaxIter', 400);
30 [theta, cost] = fminunc(@(t)(costFunctionReg(t, X, y, lambda)),
31     initial_theta, options);

```

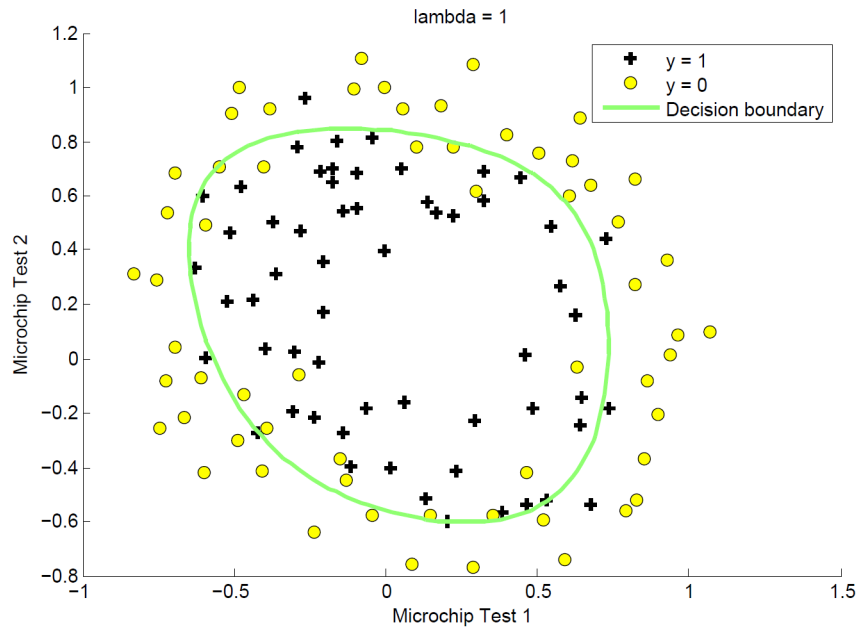


Figure 5.5: Non-linear learning problem

5.3 Multi-class classification

The goal of the TP3 and TP4 was to use logistic regression with neural network to perform automated handwritten digit recognition. We used a subset of the MNIST dataset that contains 60,000 training images. An example is plotted in the figure 5.3.

The first part of the TP3 was an extension of the TP2 to use it in a one-vs-all classification. The second part of the TP3 and the TP4 are in the next section.



Figure 5.6: Third learning problem

In this case we needed to solve a problem with 10 labels for the 10 digits. For that, we used a one-vs-all method to go back to the 2-label situation: each possible label is studied against all the others gathered. The possible label with the bigger probability becomes the prediction.

```

1 % Part 1 of TP3
2 % All the algorithms are in a vectorized form
3
4 input_layer_size = 400; % 20x20 Input Images of Digits
5 num_labels = 10; % 10 labels, from 1 to 10
6 load('ex3data1.mat'); % training data stored in arrays X, y
7 m = size(X, 1);
8
9 function [J, grad] = lrCostFunction(theta, X, y, lambda)
10     m = length(y);
11     J = 0;
12     grad = zeros(size(theta));
13
14     J = 1/m * sum(-y.*log(sigmoid(X*theta)) - (1-y).*log(1-sigmoid(
15         X*theta))) + lambda/(2*m)*(sum(theta.^2)-theta(1)^2);
16     grad = 1/m*X'*(sigmoid(X*theta)-y) + lambda/m*theta;
17     grad(1) = 1/m*sum((sigmoid(X*theta)-y).'*X(:,1));
18
19     grad = grad(:);
20
21 end
22
23 theta_t = [-2; -1; 1; 2];
24 X_t = [ones(5,1) reshape(1:15,5,3)/10];
25 y_t = ([1;0;1;0;1] >= 0.5);
26 lambda_t = 3;
27 [J grad] = lrCostFunction(theta_t, X_t, y_t, lambda_t);
28
29 function [all_theta] = oneVsAll(X, y, num_labels, lambda)
30     m = size(X, 1);
31     n = size(X, 2);
32     all_theta = zeros(num_labels, n + 1);
33     X = [ones(m, 1) X];
34
35     for c = 1:num_labels
36         initial_theta = zeros(n + 1, 1);
37         options = optimset('GradObj', 'on', 'MaxIter', 50);
38         [theta] = fmincg(@(t)(lrCostFunction(t, X, (y == c),
39             lambda)), initial_theta, options);
40         all_theta(c,:) = theta(:);
41     end
42
43 end
44
45 lambda = 0.1;
46 [all_theta] = oneVsAll(X, y, num_labels, lambda);
47
48 function p = predictOneVsAll(all_theta, X)
49     m = size(X, 1);

```

```

46     num_labels = size(all_theta, 1);
47     p = zeros(size(X, 1), 1);
48     X = [ones(m, 1) X];
49
50     A = X*(all_theta');
51     p = max(A,[],2);
52 end
53
54 pred = predictOneVsAll(all_theta, X);

```

5.4 Neural networks

The second part of the TP3 was to execute classification on neural network and the TP4 was to train the neural network.

The neural networks are a model that mix linear and non-linear steps:

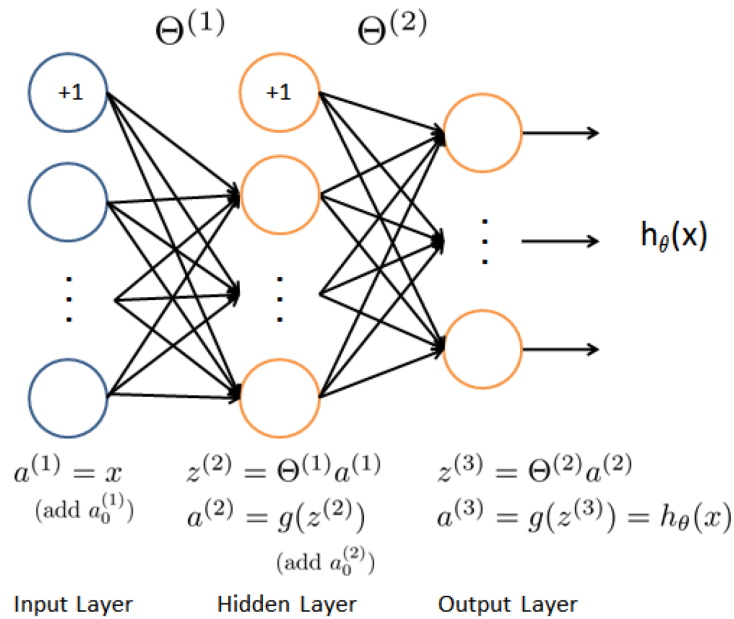


Figure 5.7: Neural Network

The figure above represents the neural network we used. The columns of circles are called layers. The first layer is called the input layer, the last is the output layers and between them, there can be different layers called hidden layers. In our exercise, there was just one hidden layer.

The θ_i are matrices and to go from one layer to the next, we needed to multiply by θ_i and to apply the sigmoid function. That corresponds to the implementation of forward propagation. At the end we have the vector $h_{\theta}(x)$.

The second part of the TP3 was the implementation of the forward propagation and the computation of the predictions:

```

1 % Part 2 of TP3 : Neural Network Classification
2
3 input_layer_size = 400;
4 hidden_layer_size = 25;
5 num_labels = 10;
6
7 load('ex3data1.mat');
8 m = size(X, 1);
9 load('ex3weights.mat');
10
11 function p = predict(Theta1, Theta2, X)
12     m = size(X, 1);
13     num_labels = size(Theta2, 1);
14
15     p = zeros(size(X, 1), 1);
16     X = [ones(m,1) X];
17     A2 = X*Theta1';
18     A2 = sigmoid(A2);
19     A2 = [ones(m,1) A2];
20     A3 = A2*Theta2';
21     A3 = sigmoid(A3);
22
23     [i,p] = max(A3,[],2);
24 end
25
26 pred = predict(Theta1, Theta2, X);

```

Once we computed forward propagation, we needed to perform a second phase called backforward propagation. By comparing the prediction to the labels, we computed an error and we made a back propagation to correct the θ matrices.

```

1 % Part of TP4: Neural network learning
2
3 load('ex4data1.mat');
4 m = size(X, 1);
5
6 load('ex4weights.mat');
7 nn_params = [Theta1(:) ; Theta2(:)];
8
9 function g = sigmoidGradient(z)
10     g = zeros(size(z));
11     sz = sigmoid(z);
12     g = sz.*(1-sz);
13 end
14
15 function [J grad] = nnCostFunction(nn_params, input_layer_size,
    hidden_layer_size, num_labels, X, y, lambda)
16     Theta1 = reshape(nn_params(1:hidden_layer_size * (
        input_layer_size + 1)), hidden_layer_size, (
        input_layer_size + 1));
17     Theta2 = reshape(nn_params((1 + (hidden_layer_size * (
        input_layer_size + 1))):end), num_labels, (

```

```

        hidden_layer_size + 1));
18
19     m = size(X, 1);
20
21     J = 0;
22     Theta1_grad = zeros(size(Theta1));
23     Theta2_grad = zeros(size(Theta2));
24
25     Y = zeros(m,num_labels);
26
27     for i = 1:m
28         Y(i,y(i)) = 1; % the label i is the column vector
29                        % (0-1-0) with a 1 at the ith position
30
31     end
32
33     % forward propagation
34     a1 = X;
35     a1 = [ones(m,1) a1];
36
37     z2 = a1*Theta1';
38     a2 = sigmoid(z2);
39     z2 = [ones(m,1) z2];
40     a2 = [ones(m,1) a2];
41
42     z3 = a2*Theta2';
43     a3 = sigmoid(z3);
44
45     h_theta = a3;
46
47     reg = sum(sum(Theta1(:,2:(input_layer_size+1)).*Theta1(:,2:(
48         input_layer_size+1)),2),1);
49     reg = reg + sum(sum(Theta2(:,2:(hidden_layer_size+1)).*Theta2
50         (:,2:(hidden_layer_size+1)),2),1);
51
52     J = 1/m * sum(sum(-Y.*log(h_theta)-(1-Y).*log(1-h_theta),2),1)
53         + lambda/(2*m)*reg;
54
55     % backforward propagation
56     delta3 = a3 - Y;
57     delta2 = (delta3 * Theta2) .* sigmoidGradient(z2);
58     delta2 = delta2(:,2:end);
59
60     Delta1 = delta2' * a1;
61     Delta2 = delta3' * a2;
62
63     Theta1_grad = 1/m * Delta1 + lambda/m*[zeros(hidden_layer_size
64         ,1) Theta1(:,2:end)];
65     Theta1_grad = 1/m * Delta2 + lambda/m*[zeros(num_labels,1)
66         Theta2(:,2:end)];
67
68     grad = [Theta1_grad(:) ; Theta2_grad(:)];
69 end

```

```

63
64 lambda = 1;
65 J = nnCostFunction(nn_params, input_layer_size, hidden_layer_size,
    num_labels, X, y, lambda);
66
67 function W = randInitializeWeights(L_in,L_out) % there is no sense to
    initialize the theta matrices with 0s
68     W = zeros(L_out, 1 + L_in);
69     epsilon_init = 0.12;
70     W = rand(L_out, 1 + L_in) * 2 * epsilon_init - epsilon_init;
71 end
72
73 initial_Theta1 = randInitializeWeights(input_layer_size,
    hidden_layer_size);
74 initial_Theta2 = randInitializeWeights(hidden_layer_size, num_labels);
75 initial_nn_params = [initial_Theta1(:) ; initial_Theta2(:)];
76
77 costFunction = @(p) nnCostFunction(p, input_layer_size,
    hidden_layer_size, num_labels, X, y, lambda);
78 [nn_params, cost] = fmincg(costFunction, initial_nn_params, options);
79
80 Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size +
    1)), hidden_layer_size, (input_layer_size + 1));
81 Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size
    + 1))):end), num_labels, (hidden_layer_size + 1));
82
83 function p = predict(Theta1, Theta2, X)
84     m = size(X, 1);
85     num_labels = size(Theta2, 1);
86     p = zeros(size(X, 1), 1);
87
88     h1 = sigmoid([ones(m,1) X] * Theta1');
89     h2 = sigmoid([ones(m,1) X] * Theta2');
90
91     [dummy,p] = max(h2,[],2);
92 end
93
94 pred = predict(Theta1, Theta2, X);

```