TRABALHO DE CONCLUSÃO DE CURSO

PERFORMANCE ANALYSIS OF PRIVATE CLASSIFICATION PROTOCOLS

Henrique Costa Jung

Brasília, dezembro de 2015

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE CONCLUSÃO DE CURSO

PERFORMANCE ANALYSIS OF PRIVATE CLASSIFICATION PROTOCOLS

Henrique Costa Jung

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro Eletricista

Banca Examinadora

FICHA CATALOGRÁFICA

JUNG, HENRIQUE COSTA
PERFORMANCE ANALYSIS OF PRIVATE CLASSIFICATION PROTOCOLS [Distrito Federal] 2015.
xvi, 39 p., 210 x 297 mm (ENE/FT/UnB, Engenheiro, Engenharia Elétrica, 2015).
Trabalho de Conclusão de Curso - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Criptografia	2. Aprendizado de Máquina
3. Processamento Seguro de Dados	4. Classificação Privada
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

JUNG, H.C. (2015). *PERFORMANCE ANALYSIS OF PRIVATE CLASSIFICATION PROTOCOLS* .
Trabalho de Conclusão de Curso, Departamento de Engenharia Elétrica, Universidade de Brasília,
Brasília, DF, 39 p.

Henrique Costa Jung
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Agradecimentos

Agradeço à família e amigos, pelo apoio que sempre me deram, e agradeço especialmente à minha mãe, por não me deixar mudar de curso.

Henrique Costa Jung

RESUMO

Este trabalho fornece um estudo detalhado sobre a performance de protocolos para Classificação Privada em Aprendizado de Máquina, relacionados a Classificação por Hiperplano. Detalhamos como o desempenho dos protocolos sofre degradação devido tanto ao aumento de classes quanto em relação ao tamanho de cada classe. Por fim, realizamos um estudo usando o banco de dados MNIST, de tamanho significativo. A conclusão a que chegamos é de que para um processamento de um banco de dados completo o custo computacional é considerável, contudo para aplicações envolvendo amostras individuais os protocolos são perfeitamente viáveis.

Palavras-chave: Criptografia, Aprendizado de Máquina, Processamento Seguro de Dados, Classificação Privada.

ABSTRACT

This work provides a detailed study about the performance of Private Machine Learning Classification protocols, related to Hyperplane Classification. We detail how the performance of protocols deteriorates due to the increase in the number of classes and the size of each class. Afterwards, we made a study using the MNIST database, of reasonable size. The conclusion we reached was that to process a full database the computational costs are severe, however for applications with individual samples the protocols are perfectly viable.

Keywords: Cryptography, Machine Learning, Secure Data Processing, Private Classification.

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Results 2
1.3 Roadmap 2

2 Preliminaries **3**
2.1 Introduction to Modern Cryptography 3
2.1.1 Algebra and Number Theory 3
2.1.2 Computational Complexity and Indistinguishability . 4
2.1.3 Definitions of Security 6
2.1.4 Two Party Computation 7
2.2 The DGK Cryptosystem 8
2.2.1 Key Generation 9
2.2.2 Encryption and Decryption 9
2.2.3 Semantic Security of the DGK Cryptosystem 10
2.3 Paillier's Cryptosystem 10
2.3.1 Security Assumptions 11
2.3.2 Encryption and Decryption 12
2.3.3 Homomorphic Properties and Self-Blinding 14

3 Private Protocols **16**
3.1 The Private Inner Product Protocol 16
3.1.1 Description of the Protocol 17
3.1.2 Correctness and Security 17
3.2 The Private Comparison Protocol 18
3.2.1 The DGK Comparison Protocol 18
3.2.2 Correctness and Security of the DGK Protocol 20
3.2.3 Private Comparison Protocol with Paillier 21
3.2.4 Correctness and Security 22
3.3 The Private Argmax Protocol 23
3.3.1 The Protocol 24
3.3.2 Correctness and Security 25

4 The Private Hyperplane Classifier 29
4.1 Private Hyperplane Protocol 29
4.2 Security 30

5 Results 31
5.1 Individual Performance 31
5.1.1 Private Comparison Protocols Benchmark 31
5.1.2 Private Hyperplane Classifier Performance 32
5.2 Classification of the MNIST Database 33
5.2.1 The MNIST Database 34
5.2.2 Linear Support Vector Machines 34
5.2.3 Overall Performance 34

6 Final Considerations 36
6.1 Conclusions 36
6.2 Future Work 37

Chapter 1

Introduction

1.1 Motivation

In recent years, privacy concerns over personal data have been growing at a great pace. Data has become a commodity, and a growing number of users of Internet services have become wary of distributing their personal data. Along with the Snowden leaks, numerous discussions have sprouted at company and government level worldwide on how much privacy an user should have. On the other hand, we have machine learning classification, an ever growing field in computer science, with innumerable applications appearing every year. Of course, private data is the realm of interesting applications, like face recognition, financial data analysis, and medical diagnosis. We can name both Apple's Touch ID and Health App as applications where the personal data of millions of users will be processed, and while those services are in theory dismissible, most users will forfeit their privacy without a second thought.

Of course, one can think of many more applications that would benefit from secure access to data. The classic example is an airport performing face recognition on the passengers: to check whether a passenger is on a terrorist blacklist, the airport must access a federal database with the faces of suspects. However, the government can't (or shouldn't) give access to classified information. At the same time, the airport shouldn't submit the faces of every traveller to the database, as without a warrant the police is forbidden to investigate the life of the citizens.

The notion that personal data will be given up easily has compelled researchers to develop methods for processing data in a private and secure way. A 2014 MIT paper [1], "Machine Learning Classification Over Encrypted Data" presents three classic machine learning classification protocols redesigned with privacy in mind, preserving the secrecy of the data of both the server (who has trained a private model) and the client (who has private data to be classified). However, [1] doesn't provide a detailed study of the performance of it's protocols, which we intend to present here.

1.2 Results

In this work, we implemented in C++ the Private Hyperplane Protocol as describe in [1]. To evaluate the performance of our implementation, we performed tests on random data, with variation in the size and shape of the samples. This way we could analyse how each subroutine weighted on the overall performance of the Protocol. We concluded that the Inner Product Protocol essentially defines the running time of the Private Hyperplane Protocol, with very little variation caused by the Argmax protocol.

Afterwards, for a proof of concept, we implemented classification using the MNIST dataset. It is a well studied machine learning problem, and we trained the model using the python library scikit-learn. This validates the model and the overall concept: although running times are much slower than in a plaintext setting, they remain altogether manageable.

1.3 Roadmap

Before implementing and benchmarking the Private Hyperplane Protocol, some study was needed. The next chapter covers the basic aspects of number theory, cryptography, and computational complexity; it is based on essentially 3 books: [6],[8],[9]. The first one is a general introduction to cryptography, and the other two explain the concepts of two-party computation. The next chapters explain the DGK and Paillier cryptosystems, the building blocks of our implementation, and are essentially rewrites of the original articles [2],[3] and [4], with conciseness and clarity in mind.

After this, we explain the Private Integer Comparison Protocol, as described in [5], and the Private Inner Product and Private Argmax, as described in [1]. Chapter 4 presents the Private Hyperplane Protocol, also described in [1]. Finally, chapter 5 presents our results, and chapter 6 the conclusions.

Chapter 2

Preliminaries

2.1 Introduction to Modern Cryptography

Cryptography can be defined as the science of hiding information, and keeping messages exchanged private. In our case, however, our interest is in controlling how much of our private data should be hidden, and how much of it can be revealed without compromising our privacy. After all, our data is only useful if it can be used.

While it may seem at first interesting to design an scheme against all kinds of attacks that a system could suffer, we face two severe problems. The first is that this kind of attack depends on the specific implementation, which we cannot know beforehand. The second is that an adversary will only try to attack us in ways we are not familiar with, and so we could not possibly design such protection beforehand. Therefore, we are concerned mainly with protecting schemes against computational attacks.

Historically, cryptographical schemes were implemented in a trial and error method, where the scheme was considered secure until someone would publicly prove it's vulnerabilities. Today, a cryptographic scheme is shown to be secure by proving it is as hard as another problem. This other problem in general is a simple mathematical question, but whose answer is believed to be hard. This way, except for errors in the implementation of the protocol, as long as the other problem remains hard, we can trust the security of the scheme.

2.1.1 Algebra and Number Theory

In this section we mean simply to give a brief surmise of the notation and symbols used in the text, specially in the chapters explaining the Paillier and the DGK encryption schemes. We will only enunciate the properties, as the proofs are easily found in most algebra and cryptographic textbooks, as is the case of our source for this section, [6].

The set of integers is represented by the symbol $\mathbb{Z}$. We say that for $a, b \in \mathbb{Z}$, a *divides* b, represented by $a|b$, if there is an unique integer $q > 0$ so that $b = q \cdot a$. If a doesn't divide b, there is an unique number r, with $0 \leq r < a$, called the *remainder* of the division of b by a, where $b = q \cdot a + r$. We also use the notation $b \bmod a$, to indicate the remainder of the division of b by a. If we say that both b and r have congruency modulo a, denoted by $b \equiv r \bmod a$, we mean that the remainder of division of b and r by a is the same.

The *greatest common denominator* of two numbers a and b, represented by $gcd(a, b)$, is the greatest number d that fulfills $d|a$ and $d|b$. Two numbers a and b are said to be *coprimes* if $gcd(a, b) = 1$. An integer p is said to be prime if its only dividers are 1 and p. This also implies that $gcd(a, p) = 1$, for $0 < a < p$.

We denote by $\mathbb{Z}_n$ the set of positive integers smaller than n, or $\mathbb{Z}_n = \{0, 1, 2, 3 \ldots n-2, n-1\}$. The set of positive integers that are smaller than n, and coprime with n is denoted by $\mathbb{Z}_n^*$. Notice that 0 is not an element of $\mathbb{Z}_n^*$.

Now, $\mathbb{Z}_n^*$ forms a closed group with the operation *exponentiation modulo n*. This means that if we pick two elements a and b in this group, $a^b \bmod n$ is also a member of this group. Euler's totient function, denoted by $\phi(n)$, gives the number of elements in the set $\mathbb{Z}_n^*$. In the case where $n = p \cdot q$, with p and q prime, $\phi(n) = (p-1) \cdot (q-1)$. For every element a in a group $\mathbb{Z}_n^*$, we have:

$$a^{\phi(n)} = 1 \bmod n \tag{2.1}$$

Carmichael's function, $\lambda(n)$, is the smallest number m that, for every $a \in \mathbb{Z}_n^*$:

$$a^m = 1 \bmod n \tag{2.2}$$

This obviously means that $\lambda(n)|\phi(n)$. Also, for $n = p \cdot q$, with p and q prime, $\lambda(n) = lcm(p-1, q-1)$. The order of an element a, $Ord_a(n)$, is the smallest number m where:

$$a^m = 1 \bmod n \tag{2.3}$$

Notice that $Ord_a(n)|\lambda(n)$. If $Ord_a(n) = \lambda(n)$, then a is a generator for the group, the group is cyclic, and set $\{a^1, a^2, a^3, \ldots, a^{Ord_a(n)}\}$ is equivalent to the set $\mathbb{Z}_n^*$.

2.1.2 Computational Complexity and Indistinguishability

To understand our definitions of security we need some basic definitions in the field of computational complexity. We will not be concerned with the full formality of computational complexity in our text, however we will use some basic concepts, retrieved from two main sources, [8] and [9].

Definition 1. *A program is said to run in Polynomial Time in n, $\mathcal{O}(p(n))$, if it always halts after a maximum of $p(n)$ steps, where $p(\cdot)$ is some polynomial. If we say it runs in Probabilistic Polynomial Time,* PPT, *we mean that the program will also makes some random choices during its computation.*

Definition 2. *A function $\mu(\cdot)$ is said to be negligible in n if for every positive polynomial $p(n)$, and sufficiently large n, we have $\mu(n) < 1/p(n)$*

Definition 3. *Let X_0 and X_1 with be two random variables with range D. We call*

$$\delta(X_0, X_1) = \frac{1}{2}\sum_{d\in D} |Pr[X_0 = d] - Pr[X_1 = d]| \tag{2.4}$$

the statistical distance *between X_0 and X_1.*

Now, suppose we have two random variables, X_0 and X_1. We want to measure the ability of an algorithm A of distinguishing between these two variables. We first sample a bit $b \leftarrow \{0, 1\}$, and than sample $x \leftarrow X_b$. We give this x to an adversary A, and it outputs a guess bit $c = A(X_b)$, representing wheter it thinks x belongs to X_0 or X_1. The algorithm A is right whenever $c = b$. Expanding this we have:

$$\begin{aligned}
Pr[c = b] &= \frac{1}{2}\left(Pr[c = b|b = 0]\right) + \frac{1}{2}\left(Pr[c = b|b = 1]\right) && (2.5)\\
&= \frac{1}{2}\left(Pr[c = b|b = 0] + Pr[c = b|b = 1]\right) && (2.6)\\
&= \frac{1}{2}\left(Pr[A(X_0) = 0] + Pr[A(X_1) = 1]\right) && (2.7)\\
&= \frac{1}{2}\left(Pr[A(X_0) = 0] + (1 - Pr[A(X_1) = 0])\right) && (2.8)\\
&= \frac{1}{2} + \frac{1}{2}\left(Pr[A(X_0) = 0] - Pr[A(X_1) = 0]\right)) && (2.9)
\end{aligned}$$

With this in mind comes the definition of advantage:

Definition 4. *The* advantage *of an algorithm A in distinguishing two distributions X_0 and X_1 is defined as:*

$$Adv_A(X_0, X_1) = 2\left|Pr[c = b] - \frac{1}{2}\right| = Pr[A(X_0) = 0] - Pr[A(X_1) = 0] \tag{2.10}$$

The *advantage* is essentially a measure of how much an algorithm perform betters than random guessing. We also have the following two properties:

$$\begin{aligned}
Adv_A(X_0, X_1) &= \delta(A(X_0), A(X_1))\\
\delta(X_0, X_1) &= max_A Adv_A(X_0, X_1)
\end{aligned}$$

A *family of random variables* $X = \{X(n)\}_{n\in\mathbb{N}}$ is a function X from non-negative integers into random variables. In our case, n will be considered the security parameter of some cryptographic scheme.

Definition 5. *We say that $X_0 = X_0(n)$ and $X_1 = X_1(n)$ are* statistically indistinguishable*, written $X_0 \equiv_s X_1$, if $\delta(X_0(n), X_1(n))$ is negligible in n. If X_0 and X_1 are not* statistically indistinguishable*, we write $X_0 \not\equiv_s X_1$. We say that X_0 and X_1 are* perfectly indistinguishable*, written $X_0 \overset{perf}{\equiv} X_1$, if $\delta(X_0(n), X_1(n)) = 0$ for all n. If X_0 and X_1 are not* perfectly indistinguishable*, we write $X_0 \overset{perf}{\not\equiv} X_1$.*

Definition 6. *We say that $X_0(n)$ and $X_1(n)$ are indistinguishable by a class of algorithms $\mathcal{A}$ if $A(X_0(n)) \equiv_s A(X_1(n))$ for all $A \in \mathcal{A}$. If $\mathcal{A}$ is the class of all* probabilistic polynomial time *algorithms, we say that X_0 and X_1 are* computationally indistinguishable*, and write:*

$$X_0 \overset{c}{\equiv} X_1 \tag{2.11}$$

Establishing the concept of *computationallly indistinguishability* is the main goal of this section. It is a weaker definition than *statistically indistinguishability*, so we have:

$$X_0 \equiv_s X_1 \Rightarrow X_0 \overset{c}{\equiv} X_1 \tag{2.12}$$

However, we will only need computational indistinguishability for our definitions of security, as in the following section.

2.1.3 Definitions of Security

With the concept of computational indistinguishability, we can define if our cryptosystems are secure or not. Essentially, we are interested in proving the *semantic security* of a cryptosystem, defined by Goldwasser and Micali in [10] . This means that knowledge of the ciphertext of a message, and the message length, does not provide any additional information about a message than the knowledge of only the message length. However, prooving a cryptosystem is secure in this definition is very hard. Luckily, this definition was proven equivalent to the definition of *Indistinguishability under Chosen Plaintext Attack (IND-CPA)*, which is much easier to prove. In the public key setting, IND-CPA works like a game played between a challenger (who has a pair of public and private keys), and an adversary who wants to break the encryption scheme denoted by π. First, the adversary has access to a polynomial amount of encryptions of any message he chooses (given that the key is public, time is the only limitation on how many encryptions the adversary has). The adversary then picks two messages m_0 and m_1, and submits them to the challenger. After this, the challenger samples a random bit b

uniformly, and computes the encryption of either m_0 or m_1, according to the bit b, and returns the ciphertext $Enc_{\pi}^{PK}(m_b)$ to the adversary. Finally, the adversary outputs his guest, a bit c. We give the following definition:

Definition 7. *A encryption scheme is said to have Indistinguishability under Chosen Plaintext Attack if, given two messages m_0 and m_1 from its plaintext space, their encryptions are computationally indintinguishable, that is:*

$$Enc_{\pi}^{PK}(m_0) \overset{c}{\equiv} Enc_{\pi}^{PK}(m_1) \tag{2.13}$$

2.1.4 Two Party Computation

A two party protocol is a random process that maps pairs of inputs to pairs of outputs. We call this a *functionality*, denoted by: $f : 0,1^* \times 0,1^* \rightarrow 0,1^* \times 0,1^*$, where $f = (f_1, f_2)$. That is, for a pair of inputs $(x, y) \in 0, 1^n$, we want to calculate the random variable $f = (f_1(x, y), f_2(x, y))$. The first party, who has x, wants the result of $f_1(x, y)$, and the second party, who has y, wants the result of $f_2(x, y)$. In an ideal world, we could simply submit the inputs to a trusted third party, who would do all the computations of f, and return the outputs to each party. As this secure third party doesn't exist, we need to develop protocols as secure as calling a third party.

The security of the protocols we will be using are based in the *static semi-honest* adversaries models. This means that one of the parties is corrupted, and wants to get as much as possible data from the other party. Traditionally, we define security so that no information is leaked to the adversaries, but this isn't a possibility here, as the adversary has access to its inputs and outputs.

Therefore, a protocol is defined as secure if a party can't learn anything more from executing the protocol than what it would learn from its own inputs and outputs. We prove this by creating a *simulation* of the protocol, and showing that the interaction with this simulator is computationally indistinguishable from the actual execution of the protocol with another party. This in turn implies that the parties learn nothing from the execution of the protocol itself. In the following definition, $view_1^{\pi}(\cdot)$ indicates what the first party sees during an execution of the protocol, x and y are the inputs of the protocol, and n is the security parameter, essentially the length of the private key. The following definition is taken from [9]

Definition 8 (Security with respect to Semi-Honest Behaviour)**.** *Let $f = (f_1, f_2)$ be a functionality. We say that protocol π* securely computes f in the presence of static semi-honest adversaries *if there exist probabilistic polynomial-time algorithms S_1 and S_2 such that:*

$$\{(S_1(1^n, x, f_1(x, n)), f(x, n))\}_{x,y,n} \overset{c}{\equiv} \{(view_1^{\pi}(x, y, n), output^{\pi}(x, y, n))\}_{x,y,n}$$
$$\{(S_2(1^n, x, f_2(x, n)), f(x, n))\}_{x,y,n} \overset{c}{\equiv} \{(view_2^{\pi}(x, y, n), output^{\pi}(x, y, n))\}_{x,y,n}$$

In our case, as the functionality f is deterministic, we need only to prove that:

$$\{S_1(1^n, x, f_1(x, n)))\}_{x,y,n} \overset{c}{\equiv} \{view_1^{\pi}(x, y, n)\}_{x,y,n}$$
$$\{S_2(1^n, x, f_2(x, n))\}_{x,y,n} \overset{c}{\equiv} \{view_2^{\pi}(x, y, n)\}_{x,y,n}$$

After proving a protocol π securely computes a functionality f, we can use that functionality while designing other protocols without concerns. For example, suppose we need to compute a functionality f_A. We decide to create a protocol π_A to compute it, but we will need to use an ideal functionality f_B as a subroutine. If we prove that an execution of protocol π_B securely computes f_B, we can prove the security of π_A using f_B, and then in a real implementation use protocol π_B as a subroutine of π_A. This is the basis for the Modular Composition Theorem, as stated and proven in [8]:

Theorem 1 (Modular Composition Theorem). *Let $f_1, f_2, \dots f_m$ be two-party probabilistic polynomial time functionalities and $\rho_1, \rho_2, \dots \rho_m$ are two-party protocols that respectively compute $f_1, f_2, \dots f_m$ in the presence of semi-honest adversaries.*

Let g be a two-party probabilistic polynomial time functionality, and Π a protocol that securely computes g in the $(f_1, f_2, \dots f_m)$-hybrid model in the presence of semi-honest adversaries. Then $\Pi^{\rho_1,\rho_2,\dots\rho_m}$ securely computes g in the presence of semi-honest adversaries.

2.2 The DGK Cryptosystem

The DGK cryptosystem was proposed by Dåmgard, Geisler and Krøigaard in [2], and was later corrected in [3]. Contrary to the Paillier Cryptosystem, the DGK does not aim to be a generic homomorphic scheme, suitable for many applications. It is actually fine tuned for performing an specific protocol, and to be very fast while performing this protocol.

As the DGK Comparison Protocol was proposed along with the encryption scheme, an unusual thing happens: a full decryption of the ciphertext is not mandatory while running the protocol. What is required is only a verification of whether or not one of the encrypted numbers decrypts to zero, which can be computed much faster. Full decryption, while faster, requires a greater computational burden at the begining of the protocol, and the choice of performing depends on the implementation.

This section explains how the keys are generated, how encryption and decryption work, and the general security; and is followed in a later chapter by a section about the DGK comparison protocol. We use as sources [2] and [3], as well as [5], in an attempt to explain the basic concepts of the cryptosystem, and its Semantic Security.

2.2.1 Key Generation

Key generation for the DGK protcol starts with a RSA modulus $n = p \cdot q$, possibly with $|n| = 1024$ bits. We then find two small primes of size t bits ([3] suggests $t = 160$), called v_p and v_q, where $v_p|(p-1)$, and $v_q|(q-1)$. Another parameter, u, is defined as an small l bits prime, with suggested value of $l = 16$. We then pick some number g at random from $\mathbb{Z}_n^*$, and try until the multiplicative order of g is uv_pv_q. This means that:

$$g^{u \cdot v_p \cdot v_q} = 1 \bmod n \tag{2.14}$$

The same thing goes for a number h, sampled at random from $\mathbb{Z}_n^*$, however we keep trying different numbers until we find one with multiplicative order of v_pv_q. The public key is the tuple $pk = (n, g, h, u)$, while the secret key is the tuple $sk = (p, q, v_p, v_q)$.

2.2.2 Encryption and Decryption

The *plaintext* space are the numbers $m \in \mathbb{Z}_u$. To encrypt a number $m \in \mathbb{Z}_u$, we pick a $2.5 \cdot t$ bits long random integer, called r, and perform the following operation:

$$Enc(m) = g^m \cdot h^r \bmod n \tag{2.15}$$

To perform decryption, we simply raise the ciphertext to v_pv_q:

$$\begin{aligned} Dec(Enc(m)) &= (Enc(m))^{v_p \cdot v_q} \bmod n \\ &= g^{m \cdot v_p \cdot v_q} \cdot h^{r \cdot v_p \cdot v_q} \bmod n \\ &= g^{m \cdot v_p \cdot v_q} \bmod n \end{aligned}$$

The last identity comes from the fact that the order of h is v_pv_q. Now, we have two possibilities. We could do a full decryption by building a table for all possible values of $g^m \bmod n$, which is viable because u is a small prime. In our protocol, however, we only want to know whether the ciphertext is an encryption of 0 or not. This can be done by simply checking if the decryption gives 1 as answer, or if:

$$g^{m \cdot v_p \cdot v_q} = 1 \bmod n \tag{2.16}$$

This obviously means that $m \equiv 0 \bmod u$. As $m < u$, we have $m = 0$. Now, suppose that p is smaller than q. For the party who generated the keys, we can be even more efficient by running decryption as:

$$Dec(Enc(m)) = Enc(m)^{v_p} \bmod p \tag{2.17}$$

2.2.3 Semantic Security of the DGK Cryptosystem

The security of the system lies in a few assumptions. First, that n is hard to factor, otherwise the system could easily be broken. The second assumption is that knowledge of g and h do not make n easier to factor. Now, let G be the group generated by g, and H the group generated by h.

Conjecture 1. *For any constant l and for appropriate choice of t as a function of the security parameter k, the tuple (n, g, h, u, x) is computationally indistinguishable from the tuple (n, g, h, u, y), where n, g, h, u are generated by the key generation algorithm, x is uniform in G and y is uniform in H.*

Proposition 1. *Under the above conjecture, the cryptosystem is semantically secure.*

Proof. We are interested in the definition of semantic security that, for a given message m, the encryption of m is computationally indistinguishable from random encryptions, i.e. $g^m \cdot h^r \bmod n \overset{c}{\equiv} g^{r_1} \cdot h^{r_2} \bmod n$. First, for $r \gg v_p \cdot v_q$, we have:

$$h^r \equiv_s y \tag{2.18}$$

Based on the conjecture above, we have:

$$h^r \equiv_s y \overset{c}{\equiv} x \tag{2.19}$$

$$h^r \overset{c}{\equiv} x = g^{r_1} \tag{2.20}$$

As x is a random element of G, and g^m a constant, we have:

$$g^m \cdot h^r \bmod n \overset{c}{\equiv} x \cdot g^m = g^{r_1} \cdot g^m = g^{r_2} \bmod n \tag{2.21}$$

$$g^m \cdot h^r \cdot h^{r_3} \bmod n \overset{c}{\equiv} g^{r_2} \cdot h^{r_3} \bmod n \tag{2.22}$$

$$Enc(m) \overset{c}{\equiv} Enc(r_2) \tag{2.23}$$

□

2.3 Paillier's Cryptosystem

Paillier's Cryptosystem was proposed by Pascal Paillier in a 1999 article [4], which is the base for all theorems, conjectures and demonstrations of this section. The security of the scheme is based on two assumptions, the *Decisional Computational Assumption* , DCRA, and the *Computational Composite Residuosity Assumption*, CCRA.

The next section explains what those assumptions are, which are in turn used to prove the security of the proposed cryptosystem. Then, we take a look at the homomorphic properties of the Paillier Cryptosystem, the *de facto* reason for using Paillier's cryptosystem.

2.3.1 Security Assumptions

We begin by explaining the security assumptions for the Paillier Cryptosystem, used while proving its security. While we do not provide the full details present in the original paper, we present enough material to justify their security and to understand what they mean. To understand the *Decisional Computational Assumption* , first we need the concept of n-th residue:

Definition 9. *A number z is said to be a n-th residue modulo n^2 if there exists a number $y \in \mathbb{Z}^*_{n^2}$ such that:*

$$z \equiv y^n mod\ n^2 \tag{2.24}$$

Determining whether a number is or is not a n-th residue modulo n^2 is believed to be a hard problem. Let's denote the problem of distinguishing n-th residues from non-residues by CR[n].

Conjecture 2 (DCRA). *The* Decisional Computational Assumption *states that there exists no polynomial time distinguisher for n-th residues from non-residues, that is,* CR[n] *is intractable.*

For the *Computational Composite Residuosity Assumption*, we need to introduce some other concepts:

Definition 10. *Carmichael's function $\lambda(n)$ is defined as the smallest number m such that:*

$$a^m \equiv 1 \bmod n \tag{2.25}$$

Where a is any number that fulfills $gcd(n, a) = 1$. For our case in particular, in which $n = p \cdot q$, we have:

$$\lambda(n) = lcm(p-1, q-1). \tag{2.26}$$

For the sake of clarity, we will refer to $\lambda(n)$ as λ for the rest of the text.

Definition 11. *For a given $g \in \mathbb{Z}^*_{n^2}$, we say that g belongs to $\mathfrak{B}$ if the order of g is a non-zero multiple of n, or for some $\alpha \in \mathbb{N}$:*

$$g^{\alpha n} = 1 \bmod n^2 \tag{2.27}$$

We should note that $g = n + 1$ is always an option, given that:

$$(1+n)^n \equiv \binom{n}{0} 1 + \binom{n}{1} n^2 + \ldots \equiv 1 \bmod n^2 \tag{2.28}$$

This definition of g gives one useful property, that of representing any $g \in \mathfrak{B}$ as an unique number of the form $(1 + n)^a \bmod n$. Also, during the setup of our system, we could randomly pick a number g and determine if $g \in \mathfrak{B}$ by checking if the following equation holds:

$$gcd\left(n, \frac{g^\lambda - 1}{n} \bmod n^2\right) = 1 \tag{2.29}$$

We call the Composite Residuosity Class Problem, Class$[n, g, w]$, as the problem of, given $w \in Z^*_{n^2}$, $y \in \mathbb{Z}^*_n$, and $g \in \mathfrak{B}$, the problem of finding the smallest m that solves the following equation:

$$w = g^m \cdot y^n \bmod n^2 \tag{2.30}$$

The original paper proves that this problem is random self-reducible over w and g, that is, it is equally hard for any w or g, and can therefore be simply called Class$[n]$. We can now define the CCRA as:

Conjecture 3 (CCRA)**.** *The Computational Composite Residuosity Assumption states that for a given $w \in Z^*_{n^2}$, $y \in \mathbb{Z}^*_n$, and $g \in \mathfrak{B}$, the problem of finding the smallest m that solves the following equation:*

$$w = g^m \cdot y^n \bmod n^2 \tag{2.31}$$

is believed to be hard. In other words, this problem, called Class[n]*, is computationally intractable.*

Finally, we should remark that the decision problems associated with DCRA and CCRA, namely CR$[n]$ and Class$[n]$, are polynomially reduced to the problem of factoring n. The proof for this is very extensive, and is covered thoroughly in Paillier's paper.

2.3.2 Encryption and Decryption

With the concepts of the previous section, we can define the basic operations for our system. For a plaintext $m < n$, $g \in \mathfrak{B}$, and $r < n$ picked at random, we define the encryption of m as:

$$Enc(m) = g^m \cdot r^n \bmod n^2 \tag{2.32}$$

On the other hand, for a ciphertext $c < n^2$, we define decryption as:

$$Dec(c) = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n)} \tag{2.33}$$

Where

$$L(u) = \frac{u-1}{n} \bmod n \tag{2.34}$$

We need to prove now that decryption of $c = Enc(m)$ yelds m, and that the encryption is safe.

Theorem 2 (Correctness)**.** *For every $c \in \mathbb{Z}^*_{n^2}$ that is the result of the encrypting of $m \in Z^*_n$, $c = Enc(m)$, decryption using equation (2) yields the correct c.*

Proof. We know that $g \in \mathfrak{B}$, therefore we can, for some $a \in \mathbb{Z}_n$, express g as:

$$g = (1+n)^a \bmod n \tag{2.35}$$

Based on that, we can precompute the bottom half:

$$\begin{aligned} L(g^\lambda) &= L((1+n)^{a\lambda}) && (2.36) \\ &= \frac{(1+n)^{a\lambda} - 1 \bmod n^2}{n} && (2.37) \\ &= \frac{1 + n \cdot a \cdot \lambda - 1 \bmod n^2}{n} = \lambda \cdot a && (2.38) \end{aligned}$$

The top half is calculated in a case by case basis:

$$L(c^\lambda) = L(g^{m \cdot \lambda} \cdot r^{n \cdot \lambda} \mod n^2) \tag{2.39}$$

From Carmichael's Theorem, any number $w \in \mathbb{Z}^*_{n^2}$ has the following property:

$$w^{n \cdot \lambda(n)} = 1 \bmod n^2 \tag{2.40}$$

Writing $g = (1+n)^a \bmod n$, we have:

$$\begin{aligned} L(c^\lambda) &= L((1+n)^{a \cdot \lambda \cdot m} \cdot 1 \bmod n^2) && (2.41) \\ &= \frac{(1 + n \cdot a \cdot \lambda \cdot m \bmod n^2 - 1)}{n} \bmod n && (2.42) \\ &= a \cdot \lambda \cdot m && (2.43) \end{aligned}$$

Finally, we can divide both bottom and top equations:

$$Dec(c) = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n)} = \frac{\lambda \cdot a \cdot m}{\lambda \cdot a} = m. \tag{2.44}$$

□

This result proves the correctness of the scheme, but we still need to prove the security of the scheme. Clearly, the knowledge of the factors of n must be kept secret, otherwise λ could be easily computed, and anyone could decrypt the messages. The scheme is also believed to be one-way, as per this theorem:

Theorem 3 (One-Wayness)**.** *The Paillier Scheme is one-way if, and only if, the Computational Composite Residuosity Assumption holds.*

Proof. The proof is rather straight forward, as breaking the scheme is by definition the Composite Residuosity Class Problem, or the problem of finding the smallest m that solves:

$$w = g^m \cdot y^n \bmod n^2 \tag{2.45}$$

And the CCRA specifically states that this problem is hard. □

On the other hand, Semantical Security is guaranteed by this theorem:

Theorem 4 (Semantic Security). *The Paillier Scheme is semantically secure if, and only if, the Decisional Composite Residuosity Assumption holds.*

Proof. Assume there are two plaintexts, m_0 and m_1, and an Adversary is given the ciphertext c for one of them. Either $c \cdot g^{-m_0} \bmod n^2$ is a n-th residue, or $c \cdot g^{-m_1} \bmod n^2$ is. Therefore, if the DCRA is false, an Adversary can easily distinguish between these two messages. □

2.3.3 Homomorphic Properties and Self-Blinding

The homomorphic properties of the Paillier Cryptosystem allow plaintext data to be processed while in an encrypted form. This way, our data can be processed by a third party, while keeping its privacy. The next two theorems explain addition and multiplication of plaintexts while encrypted. The additive property is the basis for our Secure Comparison Protocol, while the multiplicative property is the core of the Secure Inner Product Protocol.

Theorem 5. *The decryption of the product of two ciphertexts yields the sum of the corresponding plaintexts. Mathematically, let m_0 and $m_1 \in Z_n^*$ be two plaintext messages, and c_0 and c_1 their respectives encryptions. We have:*

$$Dec(c_0 \cdot c_1) = m_0 + m_1 \bmod n; \tag{2.46}$$

Proof. We have $c_0 = g^{m_0} \cdot r_0^n \bmod n^2$, and $c_1 = g^{m_1} \cdot r_1^n \bmod n^2$. Their product is:

$$\begin{aligned} c_0 \cdot c_1 &= [g^{m_0} \cdot r_0^n] \cdot [g^{m_1} \cdot r_1^n] \bmod n^2 & (2.47)\\ &= g^{m_0+m_1} \cdot (r_0 \cdot r_1)^n \bmod n^2 & (2.48)\\ &= g^{m_0+m_1} \cdot r^n \bmod n^2 & (2.49) \end{aligned}$$

We know that $Dec(g^m \cdot r^n \bmod n) = m$, for every (m, r) in $\mathbb{Z}_n \times \mathbb{Z}_n^*$, therefore

$$Dec(c_0 \cdot c_1) = m_0 + m_1 \bmod n \tag{2.50}$$

□

Theorem 6. *The decryption of a ciphertext $c_0 = Enc(m_0)$ elevated to a message m_1 yields the product of m_0 and m_1. Mathematically,*

$$Dec({c_0}^{m_1}) = m_0 \cdot m_1 \bmod n. \tag{2.51}$$

Proof.

$$\begin{aligned} c_0 &= Enc(m_0) = g^{m_0} \cdot r_0^n \bmod n^2 & (2.52)\\ c_0^{m_1} &= (g^{m_0})^{m_1} \cdot (r_0^n)^{m_1} \bmod n^2 & (2.53)\\ &= g^{m_0 \cdot m_1} \cdot r_0^{m_1 n} \bmod n^2 & (2.54)\\ &= g^{m_0 \cdot m_1} \cdot r^n \bmod n^2 & (2.55)\\ \therefore Dec(c_0^{m_1}) &= m_0 \cdot m_1 \bmod n^2 & (2.56) \end{aligned}$$

□

Another property we are interested in is Self-Blinding. This allows us to change one ciphertext into another while preserving the plaintext. This will be usefull during our Argmax protocol, where we may want to compare the same number multiple times, while keeping the server from knowing it is in fact the same number.

Theorem 7. *Any ciphertext can be publicly changed into another without altering the corresponding plaintext.*

Proof. Let c_0 be the encryption of a plantext m_0.

$$c_0 = Enc(m_0) = g_0^m \cdot r_0^n \bmod n^2 \tag{2.57}$$

We can multiply this c_0 by r^n, where r is picked at random, and get:

$$c_0 \cdot r^n = g_0^m \cdot (r_0 \cdot r)^n \bmod n^2 \tag{2.58}$$
$$c_0 \cdot r^n = c_1 \tag{2.59}$$

Decrypting c_1 and c_0 we have:

$$Dec(c_1) = Dec(c_0) = m_0. \tag{2.60}$$

□

Chapter 3

Private Protocols

This chapter presents the four basic protocols we will be using to construct the Private Hyperplane Protocol, along with their proofs of security and correctness. Here, correctness is essentially a finer detailing of what the protocol is doing, why is it doing that, and what is happening in each step of the protocol. For security, we will use the concept explained in the first chapter, that if we can create a simulation of the protocol that is computationally indistinguishable from the actual protocol, the protocol will be defined as secure, as defined by the equation:

$$\{S_1(1^n, x, f_1(x,n)))\}_{x,y,n} \overset{c}{\equiv} \{view_1^{\pi}(x,y,n)\}_{x,y,n}$$
$$\{S_2(1^n, x, f_2(x,n))\}_{x,y,n} \overset{c}{\equiv} \{view_2^{\pi}(x,y,n)\}_{x,y,n}$$

It is convenient to recall here that the concept of statistical indistinguishability ($\equiv_s$) is stronger than the concept of computational indistinguishability ($\overset{c}{\equiv}$).

3.1 The Private Inner Product Protocol

The Private Inner Product Protocol is a very simple protocol described in [1] for calculating the inner product (or dot product) of two vectors, while keeping the values of the vectors private. Suppose two parties, A and B want to calculate the inner product of their vectors, x and y respectively,each with size d, with $x = \{x_1, \ldots, x_d\}$, and $y = \{y_1, \ldots, y_d\}$. We want to calculate the following relation:

$$v = x_1 \cdot y_1 + \ldots + x_d \cdot y_d = \sum_{x=1}^{d} x_i \cdot y_i \tag{3.1}$$

The protocol is simply an implementation of this equation using Paillier homomorphic properties. We know that:

$$Enc(m_1 \cdot m_2 \bmod n) = Enc(m_1)^{m_2} \bmod n^2, \tag{3.2}$$

and

$$Enc(m_1 + m_2 \bmod n) = Enc(m_1) \cdot Enc(m_2) \bmod n^2. \tag{3.3}$$

We can easily combine these two equations and get:

$$Enc\left(\sum_i x_i \cdot y_i \bmod n\right) = \prod_i Enc(x_i)^{y_i} \bmod n^2 \tag{3.4}$$

3.1.1 Description of the Protocol

In this protocol, party B has the private key and a vector y, while party A has a vector x.

Protocol 1. Private Inner Product

1. B encrypts $y_1, y_2, \ldots, y_d$, and sends the encryptions $[y_i]$ to A.
2. A computes $[v] = \prod_i [y_i]^{x_i} \bmod n^2$.
3. A re-randomizes and outputs $[v]$.

3.1.2 Correctness and Security

Correctness

Correctness of the protocol is trivial: it holds as long as the inner product of both vectors remain smaller than n.

Security

B doesn't receive any message during the protocol, so the proof of security is straightforward. B's view is $V_B = (y, PK_P, SK_P; coins)$, where y is B's input vector, PK_P and SK_P are the public and secret keys of the Paillier scheme and $coins$ is a random tape used for the encryptions. The simulator S_B simply generates random coins $\widetilde{coins}$, which in turn come from the same distribution of the protocol. We have:

$$S_B(y, PK_P, SK_P) = (y, PK_P, SK_P; \widetilde{coins}) \overset{c}{\equiv} (y, PK_P, SK_P; coins) = V_B.$$

A's view is $V_A = (x, PK_P; coins; [y_1], [y_2], \ldots, [y_d], [v])$. We build the simulator S_A as follows:

1. Generate d encryptions of 0, $[c_1], [c_2], \ldots, [c_d]$ using the Paillier keys.
2. Calculate $\widetilde{[v]} = \prod_i [c_i]^{x_i}$.
3. Refresh $\widetilde{[v]}$, and store the random values used in the refresh operation in $\widetilde{coins}$.

4. Outputs $S_B(x, PK_P) = (x, PK_P; \widetilde{coins}; [c_1], [c_2], \ldots, [c_d]; \widetilde{[v]})$.

Both *coins* and $\widetilde{coins}$ are drawn from the same disribution, therefore they are interchangeable. Using the semantic security of the Paillier scheme, we have:

$$\begin{aligned} S_B(x, PK_P) &= (x, PK_P; coins; [c_1], [c_2], \ldots, [c_d]; \widetilde{[v]}) \\ &\overset{c}{\equiv} (x, PK_P; coins; [y_1], [y_2], \ldots, [y_d]; [v]) = V_A(x, PK_P) \end{aligned}$$

3.2 The Private Comparison Protocol

In this section we want to detail our private integer comparison protocol, as described in [5] and [1] . We need first to explain the DGK comparison protocol, more specifically the second version of the protocol described in the original paper [2]. After its explanation, and subsequent proof of security, we explain the Private Comparison protocol, which uses the DGK protocol as a subroutine. The reason for choosing the DGK protocol is a weighting between simplicity and performance, as compared to, for instance, LSIC [11].

The question that may arise is, if we already have the DGK protocol for performing secure comparison of integers, why do we need a second protocol? The answer is that the Private Comparison protocol has encrypted inputs, which are themselves the result of other operation. More important, only one of the parties holds the inputs to be compared, while the other holds the private keys.

3.2.1 The DGK Comparison Protocol

Suppose two parties A and B have two l bits numbers, x and y they want to compare privately, that is, they want to know the result of the question "Is $x \leq y$?", represented by a bit $b = [x \leq y]$. Also, they want the result to be shared among then, so that A and B each have a bit δ_A and δ_B, where $\delta_A \oplus \delta_B = b$. Lets denote by x_i the i-th bit of the binary representation of x, where i goes from 0 to $l-1$.

The intuition of the protocol is that, for comparison of two binary numbers, we are interested only on the most significant bit that differs. For instance, in the numbers $x = 11010$ and $y = 11111$, we only care that $x_2 = 0$ and $y_2 = 1$. The following equation in plaintext reflects this idea:

$$c_i = 1 + x_i - y_i + 3 \sum_{j=i+1}^{l-1} (x_j \oplus y_j) \tag{3.5}$$

If the summation on the right is not 0, that means there are differences in the more significant bits, and the result of this particular comparison does not matter. Now, if the summation is zero, we have 3 possibilities for c_i, 0, 1 and 2. $c_i = 1$ indicates that $x_i = y_i$, and therefore we are also not interested in this bit. $c_i = 2$ indicates that $x_i = 1$ and $y_i = 0$, and therefore $x > y$. $c_i = 0$ means the opposite, that $x_i = 0$, $y_i = 1$ and that $x < y$.

The DGK protocol is essentially a way of calculating all those c_i's in a secure manner, via homomorphic properties, without leaking information about the numbers, and adding an additional security parameter so that each party has a share of the answer. We denote by $[x_i]$ the encryption of the i-th bit under the DGK scheme. The DGK Protocol is as follows:

Protocol 2. The DGK Comparison Protocol

1. B encrypts each of the l y_i bits he has, and sends them in that order to A.

2. A computes the encrypted *exclusive or* of x_i and y_i by the following method:
$$[x_i \oplus y_i] = \begin{cases} [y_i] & \text{if } x_i = 0, \\ [1] \cdot [y_i]^{-1} \bmod n & \text{if } x_i = 1. \end{cases} \tag{3.6}$$

3. A chooses an uniformly random bit δ_A, and calculates $s = 1 - 2 \cdot \delta_A$.

4. For each i, $0 \leq i < l$, A computes $[c_i]$ using the following equation:
$$[c_i] = [s] \cdot [x_i] \cdot [y_i]^{-1} \cdot \left(\prod_{j=i+1}^{l-1} [x_j \oplus y_j] \right)^3 \bmod n. \tag{3.7}$$
Note that this essentially equation 3.5 above, but calculated in ciphertext space via homomorphic operations, and with the addition of the parameter s.

5. A refreshes each $[c_i]$ with a random number r_i of length $2t$ bits. This is the self blinding property of the DGK system:
$$[c_i] \leftarrow [c_i]^{r_i} \bmod n \tag{3.8}$$
After this, A sends the $[c_i]$s in random order to B.

6. B checks to see if any c_i decrypts to 0. If so, B defines $\delta_B = 1$. Else, $\delta_B = 0$.

3.2.2 Correctness and Security of the DGK Protocol

Correctness

Recall that $s = 1 - 2 \cdot \delta_A$, and that each c_i is calculated as follows, only with homomorphic operations:

$$c_i = s + x_i - y_i + 3 \sum_{j=i+1}^{l-1} (x_j \oplus y_j) \tag{3.9}$$

We have two options, party A chooses δ_A either as 1 or 0:

1. If A chooses $\delta_A = 0$ we have $s = 1$, and therefore if one of the c_i's decrypts to zero, we have $x_i = 0$ and $y_i = 1$, and $y > x$. In this case, $\delta_B = 0$, and $\delta_A \oplus \delta_B = 0$, as $y > x$.
2. If A chooses $\delta_A = 1$ we have $s = -1$, and therefore if one of the c_i's decrypts to zero, we have $x_i = 0$ and $y_i = 1$, and $x > y$. In this case, $\delta_B = 0$, and $\delta_A \oplus \delta_B = 1$, as $y \leq x$.

Security

A's view is

$$V_A = (x, PK_{DGK}, l; coins, \{r_i\}_{i=1}^{l}; \{[y_i]\}_{i=1}^{l}, \{[y_i \oplus x_i]\}_{i=1}^{l}, \delta_A, \{[c_i]\}_{i=1}^{l}).$$

Now given inputs (x, PK_{DGK}, l) we build the following simulator:

1. Generate l encrypted random bits $\{\widetilde{[y_i]}\}_{i=1}^{l}$.
2. Compute $\{\widetilde{[y_i \oplus x_i]}\}_{i=1}^{l}$ as in the protocol.
3. Pick δ_A, and calculate $\{\widetilde{[c_i']}\}_{i=1}^{l}$.
4. Pick l random values $\widetilde{r_i}$, each of length $2 \cdot t$.
5. Output

   $$S_A = (x, PK_{DGK}, l; \widetilde{coins}, \{\widetilde{r_i}\}_{i=1}^{l}; \{\widetilde{[y_i]}\}_{i=1}^{l}, \{\widetilde{[y_i \oplus x_i]}\}_{i=1}^{l}, \delta_A, \{\widetilde{[c_i]}\}_{i=1}^{l})$$

   , where $\widetilde{coins}$ are the random values used for the DGK encryptions.

As $coins$ and $\widetilde{coins}$, as well as $\{r_i\}_{i=1}^{l}$ $\{\widetilde{r_i}\}_{i=1}^{l}$ come from the same distribution, we have:

$$\begin{aligned} V_A &= (x, PK_{DGK}, l; coins, \{r_i\}_{i=1}^{l}; \{[y_i]\}_{i=1}^{l}, \{[y_i \oplus x_i]\}_{i=1}^{l}, \delta_A, \{[c_i]\}_{i=1}^{l}) \\ &\equiv_s (x, PK_{DGK}, l; \widetilde{coins}, \{\widetilde{r_i}\}_{i=1}^{l}; \{[y_i]\}_{i=1}^{l}, \{[y_i \oplus x_i]\}_{i=1}^{l}, \delta_A, \{[c_i]\}_{i=1}^{l}) \end{aligned}$$

Based on the Semantic Security of the DGK cryptosystem, we have:

$$\begin{aligned}
V_A &\equiv_s (x, PK_{DGK}, l; \widetilde{coins}, \{\widetilde{r_i}\}_{i=1}^{l}; \{[y_i]\}_{i=1}^{l}, \{[y_i \oplus x_i]\}_{i=1}^{l}, \delta_A, \{[c_i]\}_{i=1}^{l}) \\
&\overset{c}{\equiv} (x, PK_{DGK}, l; \widetilde{coins}, \{\widetilde{r_i}\}_{i=1}^{l}; \{\widetilde{[y_i]}\}_{i=1}^{l}, \{\widetilde{[y_i \oplus x_i]}\}_{i=1}^{l}, \delta_A, \{\widetilde{[c_i]}\}_{i=1}^{l}) \\
&= S_A
\end{aligned}$$

The protocol is therefore secure for party A. For party B we have $V_B = (y, SK_{DGK}, l; coins; \{[c_i]\}_{i=1}^{l}, \delta_B)$. The simulator S_B, on input (y, SK_{DGK}, l) does as follows:

1. Pick l random bit encryptions $\{\widetilde{[c_i]}\}_{i=1}^{l}$.
2. Calculates δ_B.
3. Outputs $S_B = (y, SK_{DGK}, l; \widetilde{coins}; \{\widetilde{[c_i]}\}_{i=1}^{l}, \delta_B)$, where $\widetilde{coins}$ are the random values used for the DGK encryptions.

As their drawn from the same distribution we have $coins \equiv_s \widetilde{coins}$, and from the semantic security of the DGK we have:

$$V_B = (y, SK_{DGK}, l; coins; \{[c_i]\}_{i=1}^{l}, \delta_B) \tag{3.10}$$
$$\equiv_s (y, SK_{DGK}, l; \widetilde{coins}; \{[c_i]\}_{i=1}^{l}, \delta_B) \tag{3.11}$$
$$\overset{c}{\equiv} (y, SK_{DGK}, \{\widetilde{[c_i]}\}_{i=1}^{l}, \delta_B) = S_B \tag{3.12}$$

3.2.3 Private Comparison Protocol with Paillier

The idea behind the Private Comparison Protocol is to compare the most significant bit of the two l-bit numbers a and b. Consider the number $x = 2^l + b - a$. If $b > a$, the $l + 1$ bit of x, x_{l+1}, be will equal to 1. If $b \leq a$, $x_{l+1} = 0$. We now describe the protocol, and give more insights in the proof of correctness. In this case, B has the secret key of the Paillier Cryptosystem.

Protocol 3. The Private Integer Comparison Protocol

1. A calculates $[x] = [2^l] \cdot [r] \cdot [b] \cdot [a]^{-1} \bmod n^2$
2. A picks a random number r of $l + \sigma + 1$ bits, where l is the numbers size and σ is the statistical security parameter, chosen as 100.
3. A calculates $[z] = [x] \cdot [r] \bmod n^2$, and sends $[z]$ to B.
4. B decrypts $[z]$, and calculates $d = z \bmod 2^l$.
5. A calculates $c = r \bmod 2^l$.
6. A and B run the DGK comparison protocol with inputs c and d, and obtain outputs δ_A and δ_B, where $\delta_A \oplus \delta_B = [c \leq d]$.

7. B sends $[z \div 2^l]$ and $[\delta_B]$ to A.

8. A calculates the encrypted result $[c \leq d]^{-1} = [d < c]$ as follows:

$$[d < c] = \begin{cases} [\delta_B] & \text{if } \delta_A = 1, \\ [1] \cdot [\delta_B]^{-1} \bmod n & \text{if } \delta_A = 0. \end{cases} \tag{3.13}$$

9. A calculates the encrypted result $[a \leq b] = [z \div 2^l] \cdot [r \div 2^l]^{-1} \cdot [d < c]^{-1}$

10. A sends $[a \leq b]$ to B, who decrypts and outputs $a \leq b$ in the clear.

3.2.4 Correctness and Security

Correctness

As a and b are l bits integers, $x = 2^l + b - a$ is an $l + 1$ bit integer, with $x_{l+1} = 1$ if $b > a$, and 0 otherwise. The *Private Comparison Protocol* is a way of calculating this bit securely. We can dismiss the concerns of proving correctness in the Paillier scheme ciphertext space, and deal only with plaintext, as long as we don't have carry overs modulo N. We need to make sure the range of $r = l + \lambda + 1$, remains smaller than $\log_2(N) \approx 1024$.

Since x is $l + 1$ bits long, $x \div 2^l$ is the most significant bit of x, where $\div$ denotes integer division, and we can represent x as $x = 2^l \cdot (x \div 2^l) + (x \bmod 2^l)$. We have $z = x + r$,

$$z = 2^l(z \div 2^l) + (z \bmod 2^l) \tag{3.14}$$
$$z = 2^l((x \div 2^l) + (r \div 2^l)) + ((x \bmod 2^l) + (r \bmod 2^l)) \tag{3.15}$$

We now define a bit t', where $t' = 1 \Leftrightarrow ((x \bmod 2^l) + (r \bmod 2^l)) > 2^l$. This way, $z \div 2^l = x \div 2^l + r \div 2^l + t'$. We should also notice that $t' = 0$ implies that $z \bmod 2^l = (x \bmod 2^l) + (r \bmod 2^l)$ and $t' = 1$ implies $z \bmod 2^l = (x \bmod 2^l) + (r \bmod 2^l) - 2^l$. Consequently,

$$t' = 0 \Leftrightarrow z \bmod 2^l = (x \bmod 2^l) + (r \bmod 2^l) \tag{3.16}$$
$$\Leftrightarrow z \bmod 2^l \geq (r \bmod 2^l) \tag{3.17}$$

Therefore, the bit t' is the answer of the question "Is $r \bmod 2^l$ bigger than $z \bmod 2^l$?". Finally, we come back to the equation above and get:

$$z \div 2^l = x \div 2^l + r \div 2^l + t' \tag{3.18}$$
$$\therefore x \div 2^l = (z \div 2^l) - (r \div 2^l) - t' \bmod 2 \tag{3.19}$$

Security

We begin now our proof of security of the Private Comparison Protocol. While in [1] the authors use a single encrypted bit $[t']$ as a call to a secure function evaluation of $[d < c]$, here we modify the proof by explicitly using the bits δ_A and δ_B as the two outputs of the DGK encryption protocol. We believe this proof to be more suited, as the authors of [1] do remark on using the DGK protocol instead of LSIC.

A's view is $V_A = ([a], [b], l, PK_P; r, coins; \delta_A, [\delta_B], [d < c], [b \leq a])$, where PK_P is the public key for the Paillier cryptosystem A gets from B, *coins* is the random tape used in the Paillier encryptions by A, and $\delta_A \oplus \delta_B = (d < c)$ are the results of the DGK protocol. Also, $[b \leq a]$ is the result of the comparison protocol. At first, we think of δ_A as an output of the DGK protocol, however it is obviously picked by A in the clear during the DGK protocol, and therefore it is much more fit to be viewed as an input. Therefore we have $V_A = ([a], [b], \delta_A, l, PK_P; r, coins; [\delta_B], [d < c], [b \leq a])$. Now, given $([a], [b], \delta_A, l, PK_P)$ as inputs, we build the simulator S_A:

1. Pick a random number $\tilde{r} \leftarrow (0, 2^{l+\sigma+1}) \cap \mathbb{Z}$.

2. Generate $\widetilde{[d < c]}$, $\widetilde{[\delta_B]}$ and $\widetilde{[a \leq b]}$, three Paillier encryptions of random bits.

3. Let $\widetilde{coins}$ be random coins for three Paillier encryptions.

4. Output $([a], [b], \delta_A, l, PK_P; \tilde{r}, \widetilde{coins}; \widetilde{[\delta_B]}, \widetilde{[d < c]}, \widetilde{[b \leq a]})$.

For both A's view and the simulator S_A, r and $\tilde{r}$ are taken from the same uniform distribution over $(0, 2^{l+\sigma+1}) \cap \mathbb{Z}$, and *coins* and $\widetilde{coins}$ are random tapes of same length, so

$$S_A([a], [b], \delta_A, l, PK_P) \equiv_s ([a], [b], \delta_A, l, PK_P; r, coins; \widetilde{[\delta_B]}, \widetilde{[d < c]}, \widetilde{[b \leq a]}.$$

By semantic security of the Paillier cryptosystem, we conclude with computational indistinguishability of ciphertexts:

$$\begin{aligned} S_A([a], [b], \delta_A, l, PK_P) &= \\ &= ([a], [b], \delta_A, l, PK_P; r, coins; \widetilde{[\delta_B]}, \widetilde{[d < c]}, \widetilde{[b < a]}) \\ &\overset{c}{\equiv} ([a], [b], \delta_A, l, PK_P; r, coins; [\delta_B], [d < c], [b < a]) \\ &= V_A([a], [b], \delta_A, l, PK_P). \qquad (3.20) \end{aligned}$$

3.3 The Private Argmax Protocol

The Private Argmax Protocol is a protocol that calculates in a secure manner which is the number with the maximum value in a given list. Because this protocol is a small part in a larger protocol, party A has a vector of unknown values encrypted with party B's keys, thus needing the other party to calculate the Argmax. Also, we don't want any party to learn anything else other than the Argmax, and particularly neither A nor B should learn the ordering of the values.

Suppose party A has a vector $a = a_1, a_2, \ldots, a_k$. We want to find $i_0 = argmax_i(a_i)$, that is, $a_{i_0} = max_i(a)$, and party A to output i_0.

As in the case of finding the maximum term of a list, at each iteration the current candidate and the next term of the vector are compared, but in our case we will use the Private Integer Comparison Protocol. To prevent B from learning the ordering, A simply has to apply a random permutation π to the original vector. Preventing A from learning the ordering is more complicated: at the beginning of each iteration, A adds random noise to the current candidate, max, and to the current term on the list, $a_{\pi(i)}$, and submits them to B. According to the result of the comparison, B adds it's own noise either to max or $a_{\pi(i)}$, and sends it to A, who in turn removes it's original noise, not knowing which number it received, or the result of the current comparison.

3.3.1 The Protocol

In the protocol below, party B has the Private Key of the Paillier scheme, and party A has the vector with the encrypted values. The output will be the maximum value of A's vector.

Protocol 4. Private Argmax Protocol

```
1.  A chooses a random permutation $\pi$ over $\{1, 2, \ldots, k\}$
2.  A: $[max] \leftarrow [a_{\pi(1)}]$
3.  B: $m \leftarrow 1$
4.  **for** $i = 2$ **to** $k$ **do** :
5.      Using the Private Integer Comparison Protocol, B gets the bit
        $b_i = (max \leq a_{\pi(i)})$.
6.      A picks two random integers $r_i, s_i \leftarrow (0, 2^{\lambda+1}) \cap \mathbb{Z}$.
7.      A: $[m'_i] \leftarrow [max] \cdot [r_i]$.                $\triangleright m'_i = max + r_i$
8.      A: $[a'_i] \leftarrow [a_{\pi(i)}] \cdot [s_i]$.          $\triangleright a'_i = a_{\pi(i)} + s_i$
9.      A sends $[m'_i]$ and $[a'_i]$ to B
10.     **if** $b_i$ *is true* **then**
```

```
11.            B: $m \leftarrow i$
12.            B: $[v_i] \leftarrow \mathit{Refresh}[a_i]$.
13.        else
14.            B: $[v_i] \leftarrow \mathit{Refresh}[m'_i]$.
15.        end if
16.        B sends to A $[v_i]$.
17.        B sends to A the couple $([x_i],[y_i]) = ([\bar{b_i}],[b_i])$.
18.        A: $[max] \leftarrow [v_i] \cdot [x_i]^{-r_i} \cdot [y_i]^{-s_i}$.    $\triangleright max = v_i - x_i \cdot r_i - y_i \cdot s_i$
19. end for
20. B sends $m$ to A.
21. A outputs $\pi^{-1}(m)$.
```

3.3.2 Correctness and Security

Correctness

To prove correctness, we have to show the following invariant holds: at the end of the loop for iteration i, m is the maximum of $\{a_{\pi(j)}\}_{1\leq j\leq i}$ and $a_{\pi(i_0)} = m$. If this holds, at the end of the loop iterations $a_{\pi(i_0)}$ is the maximum of $\{a_{\pi(j)}\}_{1\leq j\leq k} = a_{j\,1\leq j\leq k}$, hence $i_0 = argmax_j(a_{\pi(j)})$ and $\pi^{-1}(i_0) = argmax_j(a_j)$. At initialization, the invariant trivially holds as the family $\{a_{\pi(j)}\}_{1\leq j\leq i}$ contains only one element. Suppose the property is true for iteration $i-1$. Let us distinguish two cases:

1. If b_i is true (*i.e.*, $m \leq a_{\pi(i)}$), $max\{a_{\pi(j)}\}_{1\leq j\leq i-1} \leq a_{\pi(i)}$, as the invariant holds forthe previous iteration, and then $max\{a_{\pi(j)}\}_{1\leq j\leq i} = a_{\pi(i)}$. Then i_0 is set to i,$v_i = a'_i$ and $(x_i, y_i) = (0, 1)$. As a consequence, m is set by A to:
$$v_i - x_i \cdot r_i - y_i \cdot s_i = a'_i - s_i = a_{\pi(i)} \tag{3.21}$$
We have clearly that $a_\pi(i_0) = a_{\pi(i)} = m$ and $m = max\{a_{\pi(j)}\}_{1\leq j\leq i}$, the invariant holds at the end of the i-th iteration in this case.

2. If b_i is false $(m > a_{\pi(i)})$,$max\{a_{\pi(j)}\}_{1\leq j\leq i-1} > a_{\pi(i)}$ and
$$max\{a_{\pi(j)}\}_{1\leq j\leq i} = max\{a_{\pi(j)}\}_{1\leq j\leq i-1} = m$$
. Then i_0 is not changed, v_i is set to m'_i and $(x_i, y_i) = (1, 0)$. As a consequence,
$$v_i - x_i \cdot r_i - y_i \cdot s_i = m'_i - r_i = m \tag{3.22}$$

m is unchanged. As both m and i_0 stayed the same and $max\{a_{\pi(j)}\}_{1\leq j\leq i} = max\{a_{\pi(j)}\}_{1\leq j\leq i-1}$, the invariant holds at the end of the $i-$th iteration in this case.

Security

As this is a rather complex proof, we follow [1] very closely. We know that A's view is:

$$V_A = (\ \{[a_i]\}_{i=1}^k, l, PK_P; \pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k, coins; \\ \{[v_i]\}_{i=2}^k, \{([x_i],[y_i])\}_{i=2}^k, \pi(argmax_i(a_i))\ )$$

where *coins* are the random tapes used for encryption. To simulate A's view, the simulator S_A does the following on input $([a_1],\ldots,[a_k], l, PK_P, argmax_i(a_i))$:

1. Picks a random permutation $\tilde{\pi}$ of $\{1,\ldots,k\}$.
2. Picks $k-1$ random integers $\tilde{r_2},\ldots,\tilde{r_k}$ in $(0,2)^{l+\lambda}\cap\mathbb{Z}$.
3. Picks $k-1$ random integers $\tilde{s_2},\ldots,\tilde{s_k}$ in $(0,2)^{l+\lambda}\cap\mathbb{Z}$.
4. Generate $k-1$ random Paillier encryptions $[\tilde{v_2}],\ldots,[\tilde{v_k}]$.
5. Generates $k-1$ random bits $\tilde{b_i}$ and Paillier encryptions $([x_i],[y_i]) = ([\tilde{b_i}],[\bar{\tilde{b_i}}])$.
6. Generate a random tape for $2(k-1)$ Paillier encryptions $\widetilde{coins}$.
7. Outputs $(\{[a_i]\}_{i=1}^k, l, PK_P; \tilde{\pi}, \{\tilde{r_i}\}_{i=2}^k, \{\tilde{s_i}\}_{i=2}^k, \widetilde{coins}; \{[\tilde{v_i}]\}_{i=2}^k, \{([\tilde{x_i}],[\tilde{y_i}])\}_{i=2}^k, \tilde{\pi}(argmax_i(a_i)))$.

By the semantic security of the Paillier scheme we have:

$$\begin{aligned} V_A([a_1],\ldots,[a_k],l,PK_P) &= (\{[a_i]\}_{i=1}^k, l, PK_P; \pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k, coins; \\ &\quad \{[v_i]\}_{i=2}^k, \{([x_i],[y_i])\}_{i=2}^k, \pi(argmax_i a_i)) \\ &\overset{c}{\equiv} (\{[a_i]\}_{i=1}^k, l, PK_P; \pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k, coins; \\ &\quad \{[\tilde{v_i}]\}_{i=2}^k, \{([\tilde{x_i}],[\tilde{y_i}])\}_{i=2}^k, \pi(argmax_i a_i)) \end{aligned}$$

Given that the $\tilde{r_i}, \tilde{s_i}$ and $\widetilde{coins}$ are generated from the same distribution as r_i, s_i (uniform over $(0,2)^{\lambda+l}$) and *coins* (random tape for $2(k-1)$ Paillier encryptions), and that they are completely independent from the $\tilde{v_i}$ or π, we have:

$$\begin{aligned} V_A([a_1],\ldots,[a_k],l,PK_P) &\overset{c}{\equiv} (\{[a_i]\}_{i=1}^k, l, PK_P; \pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k, coins; \\ &\quad \{[\tilde{v_i}]\}_{i=2}^k, \{([\tilde{x_i}],[\tilde{y_i}])\}_{i=2}^k, \pi(argmax_i a_i)) \\ &\overset{c}{\equiv} (\{[a_i]\}_{i=1}^k, l, PK_P; \pi, \{\tilde{r_i}\}_{i=2}^k, \{\tilde{s_i}\}_{i=2}^k, \widetilde{coins}; \\ &\quad \{[\tilde{v_i}]\}_{i=2}^k, \{([\tilde{x_i}],[\tilde{y_i}])\}_{i=2}^k, \pi(argmax_i a_i)) \end{aligned}$$

Similarly, the distribution of $(\pi, \pi(argamax_i(a_i)))$ and $\tilde{\pi}, \tilde{\pi}argmax_i(a_i)$ is the same. As π and $\tilde{\pi}$ are independent from other parameters we have:

$$\begin{aligned}
&V_A([a_1], \ldots, [a_k], l, PK_P) \\
&\qquad\overset{c}{\equiv} (\{[a_i]\}_{i=1}^k, l, PK_P; \pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k, coins; \\
&\qquad\quad \{[\tilde{v_i}]\}_{i=2}^k, \{([\tilde{x_i}], [\tilde{y_i}])\}_{i=2}^k, \pi(argmax_i(a_i))) \\
&\qquad\overset{c}{\equiv} (\{[a_i]\}_{i=1}^k, l, PK_P; \tilde{\pi}, \{\tilde{r_i}\}_{i=2}^k, \{\tilde{s_i}\}_{i=2}^k, \widetilde{coins}; \\
&\qquad\quad \{[\tilde{v_i}]\}_{i=2}^k, \{([\tilde{x_i}], [\tilde{y_i}])\}_{i=2}^k, \tilde{\pi}(argmax_i(a_i)) \\
&\qquad= S_A([a_1], \ldots, [a_k], l, PK_P; argmax_i(a_i)).
\end{aligned}$$

Which concludes the proof for A. B's view is:

$$V_B = (SK_P, l; coins; \{b_i\}_{i=2}^k, \{[m_i']\}_{i=2}^k, \{[a_i']\}_{i=2}^k) \tag{3.23}$$

where *coins* are random coins for $(k-1)$ Paillier ciphertexts refresh. The simulator $S_B(SK_P, l)$ runs as follows:

1. Generate a random permutation $\tilde{\pi}$ of $\{1, \ldots, k\}$.
2. Set $[\tilde{a_i'}] = [i]$.
3. Run the protocol with the $[\tilde{a_i'}]$ as the input data, $\tilde{\pi}$ as the permutation, and same parameters otherwise. Let

   $$(SK_P, l; \widetilde{coins}; \{b_i\}_{i=2}, \{[\tilde{m_i'}]\}_{i=2}^k, \{[\tilde{a_i'}]\}_{i=2}^k)$$

   be B's view of this run.
4. Output $(SK_P, l; \widetilde{coins}; \{b_i\}_{i=2}, \{[\tilde{m_i'}]\}_{i=2}^k, \{[\tilde{a_i'}]\}_{i=2}^k)$.

Let $p : a_{i1 \leq i \leq k} \to 1, \ldots, k$ be the function that associates a_i to its rank among the a_i (in ascending order). Let us fix the permutation π for a while and define the following hybrids.

1. $H1 = V_B(\{[a_i]\}_{i=1}^k, l, SK_P, PK_P)$.
2. $H2 = V_B(\{[p(a_i)]\}_{i=1}^k, l, SK_P, PK_P)$.

We will show that these hybrids are statistically equal for every permutation π. As $p(\cdot)$ is a map that does not change the order of the a_i, we have that for all i, j; $a_i \leq a_j \Leftrightarrow p(a_i) \leq p(a_j)$. As a consequence, for a given permutation π, the bits b_i do not change if we replace the a_i by $p(a_i)$. Similarly, the way the a_i' and m_i' are generated for H_1 and H_2 is the same: blinding by adding random noise from $(0, 2^{\lambda+1}) \cap \mathbb{Z}$. Thus, $H_1 \equiv_s H_2$. Now, we want to show that $H_2 \equiv_s S_B(SK_P, l)$, we do not fix π anymore. Let π_0 be the permutation such that $p(a_i) = \pi_0(i)$. We can then rewrite H_2 as

$$H1 = V_B([\pi_0(1)], \ldots, [\pi_0(k)], l, SK_P, PK_P) \tag{3.24}$$

As $\widetilde{\pi}$ and $\pi \circ \pi_0$ are statistically indistinguishable, we have $H_2 \equiv_s S_B(SK_P, l)$. Recall that S_B's output is the view of B when the protocol, is run with the set $\{a_i = i\}$ as input set and $\widetilde{\pi}$ as the permutation. Hence

$$V_B([a_1], \dots, [a_k], l, SK_P, PK_P) \equiv_s S_B(SK_P, l)$$

We conclude the proof of security using modular sequential composition. We replace the ideal calls for computing the encrypted bits b_i by the provable secure Private Integer Comparison Protocol, and invoke theorem 1 to prove security in the semi-honest model.

Chapter 4

The Private Hyperplane Classifier

Hyperplane Classification is the name given to the process of classifying data using linear models, trained by algorithms such as Perceptron, Support Vector Machines, Least Squares or Fischer's Linear Discriminant. A Hyperplane is the name of a vector with one dimension less than the data, and the name Hyperplane Classification comes from the binary case, where data needs to be classified in two possible classes. The Hyperplane would then be the plane that bests divides the data in half. The difference between the several algorithms is in defining and computing this ideal plane.

In a more general setting, we have a model w that is a set of k vectors (corresponding to the k possible classes), each with size d (corresponding to the size d of each data sample). Classification in this case is done by computing the inner product of the data sample with each class of the model, and picking the class with the highest value. Mathematically, we have:

$$k^* = argmax_{i\in[k]}(\langle w_i, x\rangle) \tag{4.1}$$

Where $\langle w_i, x\rangle$ denotes the inner product of the data sample x and the coefficients corresponding to the class i of the model w.

4.1 Private Hyperplane Protocol

In this section we introduce the Private Hyperplane Protocol. It is reasonably simple, essentially using both the Private Inner Product Protocol on the data and the trained model, and the Private Argmax Protocol on the subsequent output. We can explain now the at first weird inputs of party A during the execution of the Private Argmax Protocol: the apparent paradox of not knowing the argmax of one's own data is explained as the data is the output of the Private Inner Product Protocol.

In the protocol below, a server S holds a model $w = \{w_1, w_2, \dots, w_k\}$, where each w_i has length d, and the client C has data (of size d) to be classified among the k classes. To perform the classification, the client submits it's data vector $x = \{x_1, \dots, x_d\}$, and both client and server calculate:

$$k^* = argmax_{i \in [k]} \langle x, w_i \rangle \tag{4.2}$$

Protocol 5. Private Hyperplane Classification

```
1. for $i = 1$ to $k$ do :
2.     C and S run the Private Inner Product Protocol with inputs $x$ and $w_i$.
3.     C gets the result $[v_i]$.
4. end for
5. C and S run the Private Argmax Protocol, where C has the input ciphertexts $[v_1], [v_2], \dots, [v_k]$, and C gets the result $i_0 \leftarrow argmax_{i \in [k]}(v_i)$ of the protocol.
6. C output $i_0$.
```

4.2 Security

The client's view is

$$V_C = (PK_P, x; \{[v_i]\}_{i=1}^{k}, i_0). \tag{4.3}$$

The simulator S_C, on input (PK_P, x, k^*), where $k^* = argmax_{i \in [k]} \langle x, w_i \rangle$ does the following:

1. Generate k random Paillier encryptions $[\widetilde{v_i}]$.
2. Output $(PK_P, x; \{[\widetilde{v_i}]\}_{i=1}^{k}, k^*)$.

Using the Semantic Security of the Paillier Scheme, we have:

$$V_C = (PK_P, x; \{[v_i]\}_{i=1}^{k}, i_0) \overset{c}{\equiv} (PK_P, x; \{[\widetilde{v_i}]\}_{i=1}^{k}, k^*) = S_C(PK_P, x, k^*). \tag{4.4}$$

Proving security for the Server S is a trivial case. It views only its own input, and the simulator outputs the input itself:

$$V_S = (SK_P, \{w_i\}_{i=1}^{k}) \overset{c}{\equiv} (SK_P, \{w_i\}_{i=1}^{k}) = S_S \tag{4.5}$$

Chapter 5

Results

In this chapter we present the performance results of our implementation. In the first section, the protocols are evaluated in separate, using random numbers for both the models and data samples. In the second section, we present a classical machine learning problem, the MNIST dataset, describe our classifier, Linear Support Vector Machines, and analyse the performance of this classifier in both encrypted and non-encrypted settings.

5.1 Individual Performance

To test the protocols, we performed a large number of operations with random data as a way to measure the average performance of the protocols. In this chapter, sample size refers to the size each of each sample, such as the number of pixels in an image, and class size to the number of available classes for classification.

5.1.1 Private Comparison Protocols Benchmark

We started our analysis by testing the DGK Comparison Protocol, the core subroutine of the overall scheme. To evaluate the performance of our implementation, we would take two 32 bit random numbers taken from /dev/urandom/, and perform two comparisons, using δ_A as 0 and 1. This was repeated 10,000 times to get a better average. We then repeated the whole process using 64 and 128 bits numbers.

DGK Comparison		
Number Size	Average Time [ms]	Error Rate (%)
32	17.797	0.005
64	37.230	0.012
128	83.601	0.027

As we can see in the table above, there is a linear relation between the comparison time and the size of the number we are using. If we look at the protocol this becomes clear, as there is a relation between the number size and and the number of comparisons. Fortunately, although the DGK comparison is much more expensive than a plaintext comparison, which takes about 0.08 ms in the same setting, it is still manageable.

Also, we can see there is a direct proportion between the errors rates of the protocol. Those error rates seem acceptable, however the DGK scheme is essentially deterministic, without a theoretical error rate as some other comparison schemes. Therefore, 0.005%, while small, can't be ignored. Unfortunately we don't yet know the source of these possible errors.

We then evaluated the speed of the Private Integer Comparison Protocol. In the table below, we see that the speed isn't much greater than that of the DGK comparison, because, except for some encryptions and re-randomizations, the bulk of computation corresponds to the DGK protocol. As such, the error rate also remains almost the same.

Private Integer Comparison		
Number Size	Average Time [ms]	Error Rate (%)
32	19.018	0.009
64	41.174	0.006
128	87.577	0.012

Finally, we present below a comparison between our results and [1], for 64 bits comparison. We believe the disparity to occur due to the difference in hardware, and to the mention of running the protocol in 4 different threads, which we did not implement. However, the results are still in a range of confidence.

Implementation Comparison Computational Time [ms]		
	Our's	MIT's
DGK Comparison		
32 bits	17.797	31.28
64 bits	37.230	65.18
Private Comparison		
64 bits	41.174	59.06

5.1.2 Private Hyperplane Classifier Performance

To evaluate the performance of the Private Hyperplane Classifier we performed classification on random data, while varying both the number of classes, and the amount of components of each class. While those analysis were performed, two distinct patterns emerged, which we attribute to the influence of either the Inner Product or the Argmax protocol.

By looking at how the protocols are defined, we can see that the Argmax has a direct correlation with the number of classes, and the Inner Product protocol depends on the product of the number of classes, and the number of parameters on each class. At the next table, we fixed the size of each class, but iterated on the number of classes. This way, we could analyse the effects of the Argmax protocol on the Private Hyperplane Protocol. What we see is a very linear effect on the overall performance time of the protocol.

Hyperplane Classification: Argmax Analysis
Computing Time [ms]
Number of Classes × 100

Classes	64 bits		128 bits	
	Complete	per Sample	Complete	per Sample
10	1564.5	156.45	1691	169.1
100	3947.27	39.473	8066.7	80.667
500	21904.4	43.8088	48789.7	97.580
1000	39316.8	39.317	98179	98.179

Next, we kept the number of classes fixed at 10, and then iterated on the size of each class. This way, we could evaluate the effects of the Inner Product protocol, as the Argmax protocol would be fixed. Looking at the data, we can see how little effect this has on the overall performance. 100 times more data points caused only a fourfold increase in computing time. In fact, the performance per data point actually increased.

Hyperplane Classification: Inner Product Analysis
Computing Time [ms]
10 Classes × Size of Each Class

Class Size	64 bits		128 bits	
	Complete	per Sample	Complete	per Sample
100	1514.2	15.142	1691	16.91
1000	2087.2	2.0872	2746.6	2.7466
5000	3795.45	0.75909	7640.25	1.52805
10000	6555.3	0.65553	10988.8	1.09888

From these two tables, we can draw the conclusion that the number of classes has a much bigger effect on the performance of the protocol. This isn't much of a problem, because class size tends to remain small. To many classes probably means the problem is ill defined. All and all, we can also see that the protocols run quite fast.

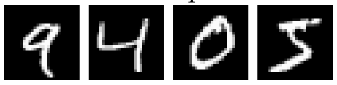
5.2 Classification of the MNIST Database

In this section, we present the performance of the protocol against a theoretical problem: handwritten digit classification. Although it has no immediate application, it servers to illustrate the performance of Private Classification, using a well established example with a well know classifier.

5.2.1 The MNIST Database

The MNIST database is a set of images representing handwritten digits, from 0 to 9. It was created by mixing samples of both U.S. Census Bureau Officers and U.S. high school students. The database is very popular in the machine learning field, being the first example presented in many courses. In [12], there is a list of the performance of several classifiers on the database.

The database consists of 70,000 images, of which 60,000 are part of the training database, and 10,000 are part of the testing database. Each sample consists of a 784 (28 x 28) pixel grid, 8-bit gray scale image. Below we present a few samples from the set:



The goal of the classifier here is, given a sample image, guess a number from 0 to 9 corresponding to it. While some more refined classifiers are able to obtain near human performance, simple classifiers without image pre-processing, as is the case here, tend to perform poorly.

5.2.2 Linear Support Vector Machines

As cited in [1], the Private Hyperplane Classifier actually corresponds to a class of linear classification algorithms. We can name Perceptron, Least squares, Fischer linear discriminant and Linear Support Vector Machines (LinearSVMs) as examples of Hyperplanes Classifiers. In our case, we have simply picked LinearSVMs, as there were no specific motivations for choosing one over the other in a validation scenario.

In a two dimensional case, SVM classification consists of finding the dividing plane with the maximum distance between the classes of the training set. This way, data points in the test set have a better chance to be correctly classified. In a multidimensional problem, the data samples are classified using a one-versus-all approach, where each class has coefficients attributed to it and we perform a inner product of the sample and each class model. The result with the highest value is thus classified.

In our case, we trained the model using python's scikit-learn library, designed specifically for machine learning use.

5.2.3 Overall Performance

After training our data, we saved the model in a txt file that was later read by our cpp program, and then performed the comparison both in a encrypted and a non-encrypted scenario, as seen below. The model was composed of floating point with 15 decimal numbers, and therefore we needed to use a 64 bit version of both the DGK and Paillier protocols.

MNIST Classification Time [ms] $28 \times 28 \times 64$ bits				
Sample Size	Plaintext		Private	
	Complete	per sample	Complete	per sample
100	593.811	5.93811	57,125.4	571.254
1000	621.731	0.62173	459,574.4	459.574
10000	876.781	0.08768	4,572,692.1	457.269

The program is obviously slow, taking nearly two hours for the complete classification of the whole set. But if we look at the individual time for each image, half a second isn't so bad for the private classification of a single image. Also, from the previous section we know that increasing the size of the image has very little effect on the computing time, so a larger image wouldn't be impossible.

Concerning the accuracy of the program, we had an error rate of 13.65% for the plaintext protocols, and of 13.68% for the encrypted protocols. While very high, this is more related to the particular problem than to the classifier. For instance, 33% of all the classification errors correspond to assigning the numbers 2, 3, 5 and 9 to the number 8. Of course, while there are many more refined classifiers, this result is very reasonable to a raw problem as this is.

Chapter 6

Final Considerations

In this final chapter, we present the conclusions of our work, and the prospects of expanding it.

6.1 Conclusions

In this work, we evaluated the performances of private protocols for integer comparison, inner products, argmax and private classifications. We concluded that while the protocols are much more expensive in computational terms than their plaintext counterparts, they are not prohibitive. Our work obtained results very similar to those in the original paper [1], corroborating the original hypothesis of usage viability.

From our data, we could also point out that the Argmax protocol, and the number of classes, have a much larger influence in the overall performance of the Hyperplane Classification than the size of each class and the Inner Product protocol. This isn't much of a problem however, as the number of classes is usually small, otherwise the problem is badly formulated. For a proof of concept, we decide to implement Linear Support Vector Machines Classifier. While Linear Support Vector Machines are not very suitable for the MNIST database in particular, they were useful given their widespread use and straightforward implementation, with classification being equivalent to the Hyperplane Classifier process.

In our implementation, we can see that it took 450ms to classify a reasonably small (28 x 28) image, which can be considered a very large time for processing an entire database. However, in a practical, case by case basis, this time can be considered minimal. Gathering suitable fingerprints and face images are lengthy process, and this further delay would be ignorable.

6.2 Future Work

There are a few options available to expand this work. Parallelism comes to mind, both in the cryptographic primitives and the classification protocols themselves. However, a preliminary survey considering the use of GPUs indicates that the communication cost would offset the computational advantage gained by splitting the communication.

The second option would be to implement the other two classifiers described in [1], Naïve Bayes and Decision Trees. This would improve the range of possible classification protocols and extend the range of problems to be treated. Additionally, with this work in hand, performance analysis against yet to be developed protocols is an obvious possibility.

Another idea is to export the implementation to a web application, or a smartphone app. Accessing a theoretical broader audience would be a better proof of concept, although we fear the focus would most likely shift to the challenges of the specific implementation instead of the subtleties of Private Classification.

A more ambitious approach would be to extend protocols with active security in mind. This would require thorough review of the literature concerning Active Security, including a reevaluation of which cryptosystems to use, and which protocols would perform faster. The choice of both the DGK and Paillier protocols would have to be taken into account, and perhaps a new, more refined cryptosystem would need to be proposed.

Bibliography

[1] BOST,Raphael et al. Machine learning classification over encrypted data. **Crypto ePrint Archive**, 2014.

[2] DÅMGARD, Ivan; GEISLER, Martin; KRØIGAARD, Mikkel. Efficient and secure comparison for on-line auctions. In: **Information security and privacy**. Springer Berlin Heidelberg, 2007. p. 416-430.

[3] DÅMGARD, Ivan; GEISLER, Martin; KRØIGAARD, Mikkel. A correction to 'efficient and secure comparison for on-line auctions'. **International Journal of Applied Cryptography**, v. 1, n. 4, p. 323-324, 2009.

[4] PAILLIER, Pascal. Public-key cryptosystems based on composite degree residuosity classes. In: **Advances in cryptology—EUROCRYPT'99**. Springer Berlin Heidelberg, 1999. p. 223-238.

[5] VEUGEN, Thijs; Improving the DGK comparison protocol. In: **Information Forensics and Security (WIFS), 2012 IEEE International Workshop** on. IEEE, 2012. p. 49-54.

[6] KATZ, Jonathan; LINDELL, Yehuda. **Introduction to modern cryptography**. CRC Press, 2014.

[7] KATZ, Jonathan; LINDELL, Yehuda. Introduction to modern cryptography. CRC Press, 2014.

[8] CRAMER, Ronald;DÅMGARD, Ivan; NIELSEN; Jesper Buus. Secure Multiparty Computation and Secret Sharing - An Information Theoretic Approach.**Book Draft**, 2012.

[9] HAZAY, Carmit; LINDELL, Yehuda. **Efficient secure two-party protocols: Techniques and constructions**. Springer Science & Business Media, 2010

[10] GOLDWASSER, Shafi; MICALI, Silvio. Probabilistic encryption. **Journal of computer and system sciences**, v. 28, n. 2, p. 270-299, 1984.

[11] VEUGEN, Thijs. Comparing encrypted data. **Multimedia Signal Processing Group, Delft University of Technology, The Netherlands, and TNO Information and Communication Technology, Delft, The Netherlands, Tech. Rep**, 2011.

[12] LECUN, Yann; CORTES, Corinna; BURGES, Christopher. **MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges**, http://yann.lecun.com/exdb/mnist/ . Retrieved 1 November 2015.