Design and implementation of an Electronic Voting system based on homomorphic tallying of votes

Treball Final de Carrera

Universidad de Lleida
Author: Andriy Bakshalov
Advisors: Josep M. Miret Biosca
Victor Mateu Meseguer

June 2014

Contents

Chapter 1

Introduction

In modern society almost everything is tending to make things in the democratic way, giving everyone a chance to make their own decisions. One of the most important processes in every democratic country is selecting a new president or the government representatives. For now, a lot of people think that this process is defrauded and the results are simply manipulated by others. This is a fear of everyone and they probably think that their vote will not change anything. This process makes people to gather in a public place and deposit their vote in the urn, and after the voting all votes are going to be counted and the result published.

With the fast and strong evolution of computer science and technology, people propose new alternatives to the traditional voting process based on computers and the internet. There is a big discussion about the most secure, faster and better way to implement them, but not only the implementation problem is holding this process, also there is a human indecision. A lot of them think that this method is not secure or it will be even worse than the traditional one, as it happened with bank accounts and internet payments, not everyone can use and trust them. Probably they can be right, because of the hundreds hacking attacks on bank systems proving that they have vulnerabilities.

For this purpose we want to implement an electronic voting process based on the homomorphic property to show that this process is easy and can be a lot more secure than the traditional one.

1.1 Objectives to be done

At this project we propose to design and implement a simulation of a real electronic voting process with Paillier encryption scheme. To realize these tasks we need to:

- Learn mathematical basis:

  - Learn about elliptic curve cryptography.
  - Learn Paillier cryptosystem.
  - Learn about homomorphic e-voting systems.

- Implementation:

  - Implement original Paillier cryptosystem and based on elliptic curves using Sage programming language.
  - Implement both cryptosystems again with a more common programming language.
  - Create a simulation of a real Electronic Voting process.

- At the end, make tests with the both implementations and compare them.

This memory contains three main parts. At the first of them we explain basic mathematical background needed to understand basic mathematics, cryptography and how Paillier cryptosystem works. The second one, explains how the work was realized. The last one, shows the results and conclusions of the work and a possible future work to be done.

Chapter 2

Mathematical Background

In this chapter we are going to explain some basic mathematical knowledge needed to understand how to work with this project.

2.1 Modular Arithmetic

Modular arithmetic is the main base of an actual public key cryptography. It is based over a positive integer N which we fix as a given, called modulo. All the numbers will have value as maximum until $N-1$ with N element converting into 0 and $N+1$ into 1.

Given two integer numbers a and b we say they are congruent modulo N and we write $a \equiv b \pmod{N}$ if N divides $a-b$.

A group is a set with a binary operation, which has an identity element, is associative and every element has an inverse. If the operation is commutative the group is called abelian.

A cyclic group is a group which has a generator g. This element can generate all the other elements by using the binary operation. According to the binary operation we can classify the groups in:

- Multiplicative group $(G, \cdot) : g \cdot \overset{n}{\cdots} \cdot g = g^n$.
- Additive group $(G, +) : g + \overset{n}{\cdots} + g = n \cdot g$.

A ring is a set with two binary operations: $(A, +, \cdot)$, where:

- $(A, +)$ is an abelian group with identity element.
- $(A, \cdot)$ satisfies the associative property.
- The binary operation $\cdot$ is distributive with respect to the binary operation $+$.

Also we can define a field which is a set with two binary operations $(K, +, \cdot)$ where:

- $(K, +, \cdot)$ is a ring whose identity element with respect to $+$ is denoted by 0.
- $(K - \{0\}, \cdot)$ is a group.

If N is a positive integer, $(\mathbb{Z}_N, +, \cdot)$, where $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$ is a ring. When N is a prime number p, the ring $(\mathbb{Z}_p, +, \cdot)$ is a field, that is, all the elements in $\mathbb{Z}_p - \{0\}$ are invertible. Moreover, we will denote by $\mathbb{Z}_N^*$ the set of invertible elements of $\mathbb{Z}_N$. It is well known that $(\mathbb{Z}_N^*, \cdot)$ is an abelian group.

2.2 Elliptic Curves

An elliptic curve over a field $\mathbb{K}$ is an algebraic curve which can be expressed by Weierstrass equation:

$$E\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \qquad a_i \in \mathbb{K}. \tag{2.1}$$

In case that, the characteristic of $\mathbb{K}$ is not 2 or 3, the equation 2.1 can be expressed in a simplified form as:

$$E\colon y^2 = x^3 + ax + b, \qquad a, b \in \mathbb{K}$$

and the discriminant Δ is defined as:

$$\Delta = -16(4a^3 + 27b^2) \neq 0.$$

The set of points expressed by $(x, y) \in \mathbb{K} \times \mathbb{K}$ which satisfy the equation of the curve with the point at infinity $\mathcal{O}$ is denoted as $E(\mathbb{K})$. The total number of points of an elliptic curve over a finite field $\mathbb{Z}_p$ is denoted as $\#E(\mathbb{Z}_p)$.

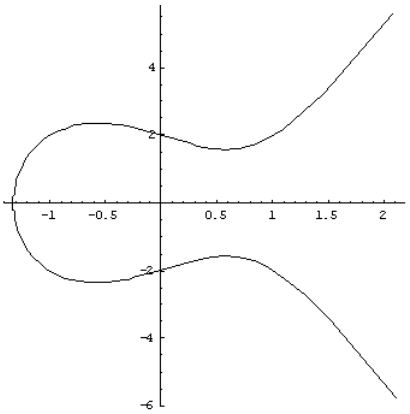

Figure 2.1: An elliptic curve over $\mathbb{R}$

At the figure 2.1 we show an example of an elliptic curve over the real field.

2.2.1 Operations with Points

Point addition

Addition of two points of an elliptic curve can be defined with a help of chord-tangent method. Moreover, with this addition operation, the set of points over $E(\mathbb{K})$ forms an abelian group with the point at infinity $\mathcal{O}$ as the identity element.

The easiest way to understand addition, is to see it geometrically. In case when we have two different points P and Q, we can connect them with a line and continue it until it cuts in a third point our elliptic curve and the opposite point will be the result. In case when we double a point P, we need to trace the tangent line at P and make the same as in the case of two different points. The figure 2.2 shows graphically the process.

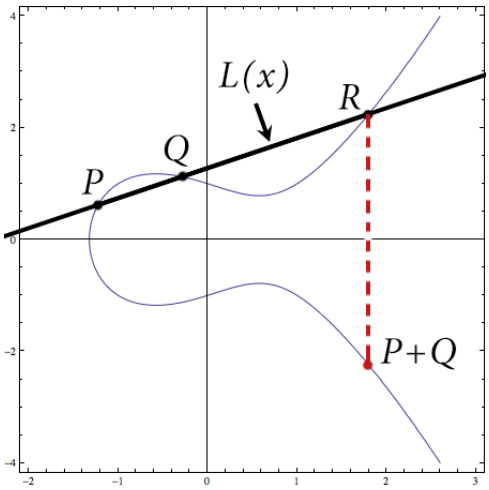

Figure 2.2: Elliptic curve point addition

In algebraic way it can be expressed as follows:

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, where P, $Q \in E(\mathbb{K})$, so the addition will be given by $P + Q = (x_3, y_3)$, where:

$$P + Q = (\lambda^2 - x_1 - x_2, (x_1 - x_3)\lambda - y_1),$$

being

$$\lambda \begin{cases} \dfrac{(y_1 - y_2)}{(x_1 - x_2)}, & \text{if } x_1 \neq x_2, \\ \dfrac{3x_2^2 + a}{2y_1}, & \text{if } x_1 = x_2 \text{ and } y_1 \neq y_2. \end{cases}$$

Point multiplication by a scalar

From point addition operation it is possible to deduce point multiplication by a scalar k. There are a lot of methods to do this operation, but one of the most optimized is the binary method. The basic idea is to transform a scalar k into the binary representation, remove the first bit and if the digit is 0 it will multiply the point by 2, and if it is 1 it will add the given point and multiply it by 2. For example, if $k = 6 \Rightarrow 110$, so we get:

$$k \cdot P = 6 \cdot P = 2 \cdot (2 \cdot P + P).$$

This method is used in elliptic curve cryptography.

2.2.2 Elliptic curves over rings $\mathbb{Z}_N$

In this case $N = p \cdot q$, where p and q are primes. For an elliptic curve E defined over a ring $\mathbb{Z}_N$ the operations are similar to the curves over a finite field, but the biggest difference that working with $\mathbb{Z}_N$ the addition operation is not possible for all the values because there is a probability to have an element which does not have an inverse because of using rings, since not all the ring elements can have an inverse. Working with big numbers N, this probability is almost zero.

Group structure of $\mathbb{Z}_N$

Let us consider the application:

$$\begin{array}{rcl} E(\mathbb{Z}_N) & \longrightarrow & E(\mathbb{Z}_p) \times E(\mathbb{Z}_q) \\ [x : y : z] & \longmapsto & ([x_p : y_p : z_p], [x_q : y_q : z_q]), \end{array}$$

defined by the projections of elements $\mathbb{Z}_N$ over $\mathbb{Z}_p$ and $\mathbb{Z}_q$. By the Chinese remainder theorem it is a bijection and as it is compatible with pseudo addition of $E(\mathbb{Z}_N)$ and with addition of $E(\mathbb{Z}_p) \times E(\mathbb{Z}_q)$ induce a group structure over $E(\mathbb{Z}_N)$ with a neuter point at infinity $[0 : 1 : 0]$. So it is possible to consider:

$$E(\mathbb{Z}_N) \simeq E(\mathbb{Z}_p) \times E(\mathbb{Z}_q).$$

Considering $N = p \cdot q$ and $d = lcm(\#E(\mathbb{Z}_p), \#E(\mathbb{Z}_q))$, the following relation holds:

$$d \cdot P = [0 : 1 : 0], \forall P \in E(\mathbb{Z}_N).$$

2.2.3 Elliptic curves over rings $\mathbb{Z}_{N^2}$

Before speaking about N^2 where $N = p \cdot q$, it is better to explain characteristics of the elliptic curves defined over the ring $\mathbb{Z}_{p^2}$, where p is prime.

Let be E an elliptic curve defined over $\mathbb{Z}_{p^2}$, so it is possible to consider this application:

$$\begin{array}{rcl} E(\mathbb{Z}_{p^2}) & \longrightarrow & E(\mathbb{Z}_p) \\ [x:y:z] & \longmapsto & [x_p:y_p:z_p], \end{array}$$

defined by the projection of elements $\mathbb{Z}_{p^2}$ over $\mathbb{Z}_p$. This application is exhaustive and its kernel is made by the set of points:

$$\mathcal{O}_k = [kp:1:0], k \in \mathbb{Z}_p,$$

and they are points at infinity of $E(\mathbb{Z}_{p^2})$. This application is compatible with pseudo addition of $E(\mathbb{Z}_{p^2})$ and $E(\mathbb{Z}_p)$. In this way, $E(\mathbb{Z}_{p^2})$ is an abelian group.

So now let us see the group structure for $E(\mathbb{Z}_{N^2})$, where $N = p \cdot q$ and p, q are primes. So the application can be considered similar to the previous as:

$$\begin{array}{rcl} E(\mathbb{Z}_{N^2}) & \longrightarrow & E(\mathbb{Z}_{p^2}) \times E(\mathbb{Z}_{q^2}) \\ [x:y:z] & \longmapsto & ([x_{p^2}:y_{p^2}:z_{p^2}],[x_{q^2}:y_{q^2}:z_{q^2}]). \end{array}$$

By Chinese reminder theorem it is possible to prove that this is a bijective application, so it induces a structure of group in $E(\mathbb{Z}_{N^2})$.

Considering $N = p \cdot q$ and $d = lcm(\#E(\mathbb{Z}_p), \#E(\mathbb{Z}_q))$. The following relation holds:

$$d \cdot N \cdot P = [0:1:0], \forall P \in E(\mathbb{Z}_{N^2}).$$

Elliptic curve point classification over $\mathbb{Z}_{N^2}$

The points of an elliptic curve E over $(\mathbb{Z}_{N^2})$, $N = p \cdot q$ can be separated into tree groups:

- Points at infinity: $\mathcal{O}_k = [kN:1:0], k \in \mathbb{Z}_N$.
- Points semi-infinity: $[x:y:z] \in E(\mathbb{Z}_{N^2})$, where $\gcd(z, N) = p$ or q.
- Affine points: $[x:1:z] \in E(\mathbb{Z}_{N^2})$.

In this case, the method of adding points remains the same, by the rule of chord and tangent, but only in case when exist inverses modulo N^2 necessary for the calculation. So the changes to the algebraic operation of adding and doubling will be:
When $x_1 \neq x_2$ the sum of (x_1, z_1) and (x_2, z_2) is given by (x_3, z_3):

$$\begin{array}{rcl} x_3 & = & x_1 + x_2 + (z_1 - \lambda x_1)(2a\lambda + 3b\lambda^2)/(1 + a\lambda^2 + b\lambda^3), \\ z_3 & = & \lambda(x_3 - x_1) - z_1, \\ \lambda & = & (z_1 - z_2)/(x_1 - x_2). \end{array}$$

In case when $x_1 = x_2$ the doubling of (x_1, z_1) is:

$$\begin{array}{rcl} x_3 & = & 2x_1 + (z_1 - \lambda x_1)(2a\lambda + 3b\lambda^2)/(1 + a\lambda^2 + b\lambda^3), \\ z_3 & = & \lambda(x_3 - x_1) - z_1, \\ \lambda & = & (3x_1^2 + az_1^2)/(1 - 3bz_1^2 - 2ax_1z_1). \end{array}$$

Where a, b, x_1, x_2, x_3, z_1, z_2, z_3, $\lambda \in N^2$.

With this operation it is possible to show that:

$$\mathcal{O}_k + \mathcal{O}_h = \mathcal{O}_{k+h} = [(k+h) \cdot N : 1 : 0].$$

2.3 Cryptography

From the times of the old Egypt until today, people always wanted to hide their secrets from others. The first methods to hide them were simply reordering letters in the words or changing a letter for the next one from the alphabet, but those methods were not secure to keep the information in secret from the third party members.

One of the improvement was Caesar cipher, where letters were changed by a fixed number of positions further down the alphabet, and this number was secretly passed to the destination, so he is the only one who could decrypt and obtain the original message.

With the birth of computers and information technology changed everything, because of the calculation power, so all the cryptography had to make a greater improvement of their complexity. From DES (Data Encryption Standard) a modern symmetric cryptosystem to the public key cryptography and, in particular, elliptic curve cryptography. There is a big step which offers new advantages.

In the modern society, cryptography is focused over the internet and helps users to interchange data securely. The basic model of how this process works:

- Sender encrypts a message and sends the result to the receiver.
- The encryption is transmitted by an insecure channel.
- The receiver receives the encrypted message and decrypts it to obtain the original message.

So this is why we need cryptography, if we want to send our message by some insecure way, first of all we need to transform it in something not understandable for others and if they pretend to steal the secret, they will have no idea how to transform it into a readable text.

By the flow of time, nowadays we have different types of cryptosystems:

Symmetric cryptosystems : are the simplest cryptosystems. They are based on that both users have the same key and it works in two ways. The most common are:

- DES (Data Encryption Standard) [11]: this algorithm based on 64 bits keys, where 56 bits are for the key and other 8 to check and correct

errors. Nowadays this algorithm is not a standard any more because it was broken and 56 bits keys are not secure for the modern computers.

- AES (Advanced Encryption Standard) [9]: it is a symmetric algorithm which works with keys of 128, 192 and 256 bits. AES became effective as a federal government standard on May 26, 2002 after approval by the Secretary of Commerce.

- IDEA (International Data Encryption Algorithm) [5]: it is based on simple operations as: addition, multiplication and XOR with block sizes of 64 bits and key size 128 bits.

Asymmetric cryptosystems (public key cryptography) : are more complex than the symmetric because they use different keys for encryption and decryption. This way, these cryptosystems are more secure than the symmetric ones.

Some example of them are:

- RSA [14]: this is one of the most used cryptosystems over the internet. The difficulty stands on factoring the product of two large prime numbers p and q. The main problem is the use of computational easy numbers that can be calculated fast, as the system does not use any random element. There are a lot of attacks for this algorithm, but as prime numbers increase their size, more difficult becomes to break it.

- ElGamal encryption system [3]: is an asymmetric key cryptosystem which is based on DiffieHellman key exchange and it is used at GNU Privacy Guard software. Encryption of this method is defined over a cyclic group G and the security depends on the difficulty of computing the discrete algorithm in G. ElGamal encryption is probabilistic, meaning that a single plaintext can be encrypted to many possible ciphertexts, with the consequence that a general ElGamal encryption produces a 2:1 expansion in size from plaintext to ciphertext. Moreover, the encryption needs two exponentiations, but they are independent and the decryption needs only one.

- Paillier cryptosystem [12]: is one of the asymmetric systems which works with two prime numbers, as in case of RSA, but uses a cyclic group of order N^2, where $N = p \cdot q$. One of the important characteristics of this system is the use of a random element and the homomorphic property which is going to be explained in the next chapter.

A special type of cryptographic functions are **hash functions** which are functions that compress, meaning that the output will be shorter than the input length. Often, these functions take a random length input and convert it into one whose length is a fixed number, like 160 bits. Hash functions are

used in many parts of the cryptography, as digital signatures. There are many different types of them, with different security properties, the main ones are:

- One way functions. It is not possible return hashed value to its original state.
- Collision free. It is almost impossible to find two different messages with the same hash value.
- Efficiency. It is very easy to calculate hash values.
- Uniform. Messages with arbitrary length will produce fixed length hash.

The common used Hash functions are:

- MD5 (Message Digest 5) [13]: it was an improvement of MD4 algorithm and became a widely used algorithm which produced 128 bit hash values. By the year 2004 it was showed that it is not collision resistant and it was proposed to use SHA-1 for more security.
- SHA-2 (Secure Hash Algorithm - 2) [10]: is an improvement to SHA-1 hash function and provides digesting of 256 bits messages to the hash values of 256, 384 or 512 bits. This hash is more secure than MD5, but it is more difficult to calculate.

Homomorphic property

Some cryptosystems have a special ability called homomorphic property, where it is possible to make some arithmetical operations with the encrypted text without decrypting, such as addition or multiplication. When the system has this property, it can be used to multiply a ciphertext and after the decryption the result will be the addition of both original messages. This is supported by Paillier cryptosystem which is going to be explained next with more detailed specification of the homomorphic property.

2.3.1 Paillier Cryptosystem

Paillier cryptosystem is a probabilistic algorithm with a public key encryption scheme. The security is based on computing the n-th residue which is believed to be computationally difficult.

In order to define this problem, given an integer $N = p \cdot q$, with p and q prime numbers, and an element $g \in \mathbb{Z}^*_{N^2}$ of order multiple of N (being $\#\mathbb{Z}^*_{N^2} = N(p-1)(q-1)$), Paillier considered the function:

$$\begin{array}{rcl} F_g : \mathbb{Z}^*_N \times \mathbb{Z}_N & \longrightarrow & \mathbb{Z}^*_{N^2}, \\ (r, m) & \longmapsto & r^N g^m \pmod{N^2}. \end{array}$$

This is a bijective function, so there exists an inverse function. Thus, given $c \in \mathbb{Z}_{N^2}^*$ there exists a unique $m \in \mathbb{Z}_N$ such that there is a unique $r \in \mathbb{Z}_N^*$ such that:

$$F_g(r, m) = r^N g^m = c.$$

This element $m \in \mathbb{Z}_N$ is called the N-residuosity class of c and it will be denoted by:

$$m = ||c||_g.$$

Given $c, g \in \mathbb{Z}_{N^2}^*$, the problem of finding $m \in \mathbb{Z}_N$, such that $m = ||c||_g$ is a computationally hard problem called the composite residuosity class problem). Nevertheless, it has been shown that from the knowledge of the factorization of $N = p \cdot q$ we can deduce the integer $m \in \mathbb{Z}_N$.

Initialization

As a first step to use this system we need to generate public and private information, the steps are:

(1) Choose two big random prime numbers p and q satisfying

$$\gcd((p-1), (q-1)) = 1.$$

(2) Calculate $N = p \cdot q$.

(3) Calculate $\lambda = lcm(p-1, q-1) = (p-1)(q-1)$.

(4) Generate a random number $g \in \mathbb{Z}_{N^2}^*$.

(5) We need to be sure that N divides the order of g and insure that it has a multiplicative inverse:

$$\mu = L(g^\lambda \pmod{N^2})^{-1} \pmod{N},$$

where the function L is defined as:

$$L(u) = \frac{u-1}{N}.$$

Completed those steps it is possible to define public information (N, g) and the private values (λ, μ) which is going to keep in secret.

Encryption

The message to be sent will be denoted as m, where $m \in \mathbb{Z}_N^*$. To encrypt the message it is needed to make these steps:

(1) Generate random element r, which is $\in \mathbb{Z}_N^*$.

(2) Calculate encrypted text using the function:

$$c = E(m, r) = g^m \cdot r^N \pmod{N^2}.$$

This cryptosystem uses a random element r which makes possible that two different encrypted messages are the same when decrypted. This makes that if somebody gets an access to the channel where your messages are passing, he can not even guess if those messages are the same or not.

Decryption

After the receiver got the encrypted message $c \in \mathbb{Z}_{N^2}^*$, he needs to decrypt it using his private keys (λ, μ):

$$D(c) = L(c^\lambda \pmod{N^2}) \cdot \mu \pmod{N}.$$

Notice that:

$$D(c) = D(E(m, r)) = m.$$

Indeed,

$$D(E(m, r)) = D(g^m \cdot r^N \pmod{N^2}) = \frac{(g^m \cdot r^N)^\lambda - 1}{g^\lambda - 1} = \frac{(g^m)^\lambda - 1}{g^\lambda - 1}.$$

Now, taking into account that

$$t^\lambda = 1 \pmod{N},\ \forall t \in \mathbb{Z}_{N^2}^*,\ \lambda = (p-1)(q-1),$$

we get

$$g^\lambda = 1 + k \cdot N, k \in \mathbb{Z}_N, \text{ and } (g^\lambda)^m = 1 + m \cdot k \cdot N \pmod{N^2}.$$

From this, it follows that

$$D(E(m, r)) = \frac{m \cdot k \cdot N}{k \cdot N} = m.$$

Homomorphic property

This cryptosystem offers important homomorphic property which gives a set of advantages as:

- Multiplication of two cyphered texts will be decrypted as the sum of the original messages, like this:

$$D(E(m,r)\cdot E(m',r')) \pmod{N^2} \;=\; m+m' \pmod{N}.$$

- Cyphered text elevated to the power k can be decrypted as a multiplication, like this:

$$D(E(m,r)^k) \pmod{N^2} \;=\; k\cdot m \pmod{N}.$$

2.3.2 Paillier with Elliptic Curves

In this section we present a version of Paillier cryptosystem using elliptic curves [4].

Initialization

As in the previous section, we have to generate a set of values for encryption and decryption processes.

(1) Choose two big random prime numbers p and q satisfying

$$\gcd((p-1),(q-1)) = 1.$$

(2) Calculate $N = p\cdot q$ and N^2.

(3) Choose two random numbers a and $b \pmod{N}$ and generate an elliptic curve E over $\mathbb{Z}_{N^2}$.

(4) Calculate the cardinal of the elliptic curve E over the field $\mathbb{Z}_p$ and over the field $\mathbb{Z}_q$.

(5) Calculate $d = lcm(\#E(\mathbb{Z}_p), \#E(\mathbb{Z}_q))$.

(6) Generate a random point Q' on the elliptic curve E over the ring N^2 and calculate Q, where $Q = N\cdot Q'$.

The problem of this system is possibly laying in the operations with points where the coordinates are multiples of p or q because of using N^2, so it will be impossible to find the inverse of the element and it will crush the system and all the process. This is more probable using low values for the primes p

and q, but as range grows, the probability grows down. In conclusion, not always the obtained values are good to use.

After obtaining all the necessary information it is possible to define the public information (N, Q, a, b) and the private key d.
For more information about this cryptosystem consult [4].

Encryption

Let us denote the message to be encrypted as $m \in \mathbb{Z}_N^*$, and consider the point $P_m = [m \cdot N : 1 : 0]$ on the elliptic curve E over $\mathbb{Z}_{N^2}$.
To encrypt the message it is needed to follow these steps:

(1) Generate random element r, which $\in \mathbb{Z}_N^*$.

(2) Calculate the points $r \cdot Q$, P_m on the curve over $\mathbb{Z}_{N^2}$ and add them to obtain the encrypted point:

$$S = r \cdot Q + P_m.$$

Decryption

To decrypt the ciphertext S it is needed to use the private key $d = lcm(\#E(\mathbb{Z}_p), \#E(\mathbb{Z}_q))$.

(1) Calculate the point $S' = d \cdot S$ on the curve E over $\mathbb{Z}_{N^2}$. Note that $S' = d \cdot P_m = [d \cdot m \cdot N : 1 : 0]$.

(2) From the point S', dividing the first coordinate by N we obtain $x = d \cdot m$.

(3) Calculate the original message m multiplying x from the step (2) by d^{-1}:

$$m = x \cdot d^{-1} \pmod{N}.$$

Chapter 3

Electronic Voting

In our democratic society one of the most important things is freedom to choose something better for you. In every democratic country the process of voting is a very important event where people choose new government to rule their country for the next years and make it better in all the ways.

As probably everyone knows, today not all the countries can afford a secure and democratic election process. Sometimes, coercion of the voters forces people to vote for a determinate candidate, that happens quite frequently in poor countries.

With such a powerful progress of the information technologies, people start to propose to elevate this voting process to a new level, a level of Electronic Voting (E-voting), where voters can be sure that their vote will not be manipulated or other people would not know which candidate did they vote. The main reasons of implementing this type of voting are:

a) It can increase the security in the way that it will be more difficult to relate the voter with the emitted vote.

b) People can access from their houses, so there is no need to prepare voting houses, employ people and waste money on the preparation.

c) Every voter will have the possibility to check if his emitted vote was counted correctly.

Electronic voting makes the voting process simpler for the government and for the citizens. Of course it has different internal stages like:

- Preparation.
    - Publishing the election. During this process the list of candidates along with the public information and the software/hardware requirements for voting will be announced.

  - Registry of voters. At this step it is important to collect the data from the participants at election, to prepare their identification and avoid errors.

- Voting.

  - Identification of the voter. Every voter needs to introduce his identification information to be able to participate in the election.
  - Candidate selection. When the user is successfully identified he will be able to see the list of candidates and make his choice.
  - Sending the vote. After the voter has chosen his candidate he will send his vote to the server.
  - Validation of the vote. When the server receives the vote, it has to check if all the data is correct and then it can be added.

- Final results.

  - Transferring the votes. When the voting time has ended all the votes have to be transferred to a trusted party where they are going to be decrypted.
  - Permutation. At this step, the relation between the voter and his ballot is broken.
  - Decrypting and adding the votes. Recover the initial message and add the vote to the voted candidate.
  - Checking the results. After all the job is done, there is a need to check if all the data was proceeded correctly.
  - Publishing the result. After all the checkings, the results can be safely published.

Those are the common steps of electronic voting. There are a lot of different cryptosystems which can offer an easier implementation of the voting step as they have the homomorphic property so there will be no need to decrypt each vote one by one. It will happen with Paillier cryptosystem as it is detailed in Chapter 4.

3.1 Paradigms

In order to guarantee that the system satisfies all security requirements there are different cryptographic techniques.

3.1.1 Mixnet protocols

This protocol is the closest to traditional model of voting. The voter chooses his candidate, and encrypts the message with his choice. Then the ballot is sent to the server which will retain all the encrypted votes. After the voting process, ballots will be mixed and cyphered again, then opened one by one and tallied. This way, the protocol breaks any relation between the votes and their emitters.

The mixing process is a low cost operation and the difficulty grows in a linear way, but the difficulty of repeating the cyphering process depends on the amount of votes and the cryptosystem key length, so the cost grows significantly. But what if, new cyphering creates n new votes and changes them by the existent ones. The solution for this problem are the proofs of correctness.
For more information see [8].

Proofs of correctness

These proofs are used to show that the messages have not been changed during the mixing. There are different types: interactive and non-interactive. At the interactive ones, it is needed to interchange the information between the user and the verifier. This limits the number of people who can carry out them in terms of voting. The other ones, do not need any interaction, the user sends all the information to the verifier and the testing information is public. This way, it verifies that the mixing process was correct and does not waste the time on waiting for a response.
More information about the proofs can be found at [6].

3.1.2 Homomorphic tally protocols

Homomorphic voting process require cryptosystems which are supporting the homomorphic property. This is useful in electronic voting because we can join the votes when they are encrypted and, at the end, only make one decryption of the resulting sum. This way, it decreases the number of decryptions and the aggregated result can not be directly linked with any voter.

This type makes the decryption process easier and faster, but there is a problem of receiving corrupted votes. If they are added to the sum of votes it can produce the alteration of the final result. For this reason there exist proofs which check that a message lies in a set (MLS proofs) without decrypting it.
For more information about this protocol see [1].

MLS Proofs

One of the important parts of electronic voting is to demonstrate that the incoming vote encrypts a valid message and this is not some intent of breaking the system or manipulating the election. To avoid this type of insecurity it needs to implement a set of tests called MLS proofs.

These tests can be carried out over different cryptosystems. But, in this case, it has been implemented over Paillier, since it has the homomorphic property.

For this test, there will be two participants: the user and the verifier. The user has to prove to the verifier that encrypted text:

$$c = g^{m_i} \cdot r^N \pmod{N^2} \tag{3.1}$$

encrypts a message belonging to $S = \{m_1, m_2, \ldots, m_p\}$, where m_i represents an original message.

- The user generates $\rho \in \mathbb{Z}_N^*$. He also generates $p-1$ random values $\{e_j\}_{j \neq i} \in \mathbb{Z}_{\mathbb{N}}$ and $p-1$ values $\{v_j\}_{j \neq i} \in \mathbb{Z}_N^*$.
- The user calculates:
$$u_i = \rho^N \pmod{N^2}$$
and
$$\{u_j = v_j^N (g^{m_j}/c)^{e_j} \pmod{N^2}\}_{j \neq i}.$$
- The user generates $e_{shall} = H(u_1, \ldots, u_p)$ where H represents a hash function, and calculates:
  - $e_i = e_{chall} - \sum_{j \neq i} e_j \pmod{N}$.
  - $v_i = \rho \cdot r^{e_i} \cdot g^{(e_{shall} - \sum_{j \neq i} e_j) \div N} \pmod{N^2}$, where $\div$ is a quotient of the integer division.
- The user sends $\{u_j, v_j, e_j\}_{j \in \{1,\ldots,p\}}$ and e_{chall} to the verifier.
- The verifier checks that $e_{chall} = \sum_j e_j \pmod{N}$.
- For the last one, the verifier checks that $v_j^N = u_j (c/g^{m_j})^{e_j} \pmod{N^2}$ for every $j \in \{1, \ldots, p\}$.

More information about the proofs can be found at [7].

3.1.3 Blind Signature-based protocols

A challenge for electronic voting is to guaranty the privacy for all the participants, ensuring that their votes can not be related to them. The idea of this methodology consists in that the person will apply a blind factor for the message m and send it to the authentication authority which will sign

the received message and return it to the sender. After this, the person will remove the blind factor from the signed message and sent it to the server through an anonymous channel. When the voting period is concluded, votes are decrypted and tallied.
For more information see [2].

3.2 Security Requirements

The most challenging part of e-voting is to show that this type of voting process is secure.

The most important properties which every Electronic voting process has to have are:

- Confidentiality. No one knows who voted in favour of whom.
- Uniqueness. Only one vote for each voter.
- Authorization. Only the authorized person can make his vote.
- Precision. All the votes have to be checked for the correctness to avoid the errors.
- Partial results. No information can be shared before the final result was published.
- User check. Every participant can check if his vote was received and tallied.

In the voting process it can be described three roles:

- Voter. Each person which will choose their candidate and send the encrypted and signed message to the server.
- Polling Station. A server which will collect all the votes and publish the result of the process.
- Trusted Authority (TA). It is responsible of storing the private key and decrypting the votes sent by the Polling Station.

Chapter 4

Electronic Voting with Paillier cryptosystem

After explaining the basics about electronic voting it is possible to explain how to design and implement a voting process with the Paillier cryptosystem.

First of all, Paillier cryptosystem is homomorphic, so the Polling Station (server) will receive the votes and check them with the MLS proofs. After passing the proof, it will be added to the sum of all the votes. When the election time is over, the server will not accept new incoming votes. It will send the encrypted sum to the TA, which will decrypt it with the private key and return the decrypted message to the server. Then it will translate the result into the votes for each candidate and publish them. Let us take a look on each identity.

Voter will receive a list of possible candidates to vote and have the possibility to choose one of them. When the candidate is chosen, the voter sends his vote to the server, so the client part will take his message and encrypt it using the Paillier cryptosystem with the public information. After that, the vote is signed and sent to the polling station for validation and further work.

Polling Station plays the role of the server and contains the list of candidates for voting. In this case, the server will only gather all the received votes and make a sum of them, but only if they passed the MLS proofs. When the server will receive the order of stopping the voting process it will reject all the new incoming messages and send the sum of votes to the trusted authority. After receiving the message from the trusted authority it will relate each candidate with his obtained votes and publish the result of the voting process.

TA only accepts messages from the server and its function is only to decrypt the incoming messages from the polling station with the private key proving that it has been decrypted using the appropriate private key.

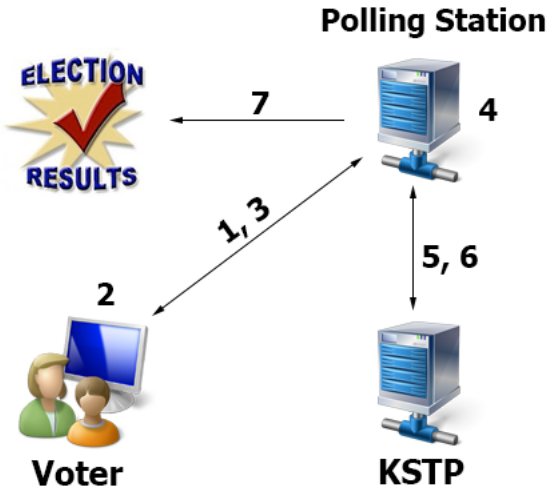

Figure 4.1: Electronic Voting design

This way, it is possible to see that this structure is proposing high security level, as the vote is travelling fully encrypted through the public channel, so if nobody has the private key it is impossible to know which is the original message. When the vote comes to the server it is checked to be correct and added to the total sum of votes but it is never decrypted. This way, the relation between the voter and his vote is totally lost. The polling station has no knowledge about how to decrypt the accumulated sum, so even if somebody attempt to find this information it is impossible to decrypt it without knowing the private key.

Figure 4.1 shows how this system can be represented graphically.

1) The Polling Station send to the voter the list of candidates.

2) A Voter select his candidate.

3) A Voter send his encrypted vote to the Polling Station.

4) A Votes are added to the total at the Polling Station.

5) The Polling Station send encrypted final result to the TA.

6) The TA returns decrypted result to the Polling Station .

7) The Polling Station publishes the result of the election.

4.1 Referendum voting

With Paillier, it is possible to make a referendum voting as it contains only the votes "accept" and "reject", so we can say that the vote "accept" will be a message with value 1 and "reject" with value 0. Thus, when the voter will choose his selection and send it to the polling station, internally the system will encrypt it using Paillier crptosystem and send it to the server which will add it to the total sum using the homomorphic property.

In this way the vote of the voter i will be encrypted as

$$V_i = r_i Q + P_{m_i},$$

where $P_{m_i} = [m_i N : 1 : 0]$ and $m_i \in \{0, 1\}$.

To check the correctness of the message we use the MLS proofs and make sure that V_i encrypts a message in range $m \in \{0, 1\}$.

The server will collect the votes and add them to the total sum, obtaining:

$$\sum_{i=1}^{n} V_i,$$

where n is the total number of participants in the electoral roll. Recalling from the properties of the elliptic curves over N^2 we have that it will contain the addition of the points at infinity, so the result will keep the addition of the votes:

$$\sum_{i=1}^{n} V_i = (\sum_{i=1}^{n} r_i) \cdot Q + P_{m_1+\cdots+m_n},$$

where $P_{m_1+\cdots+m_n} = [(m_1+\cdots+m_n)\cdot N : 1 : 0]$ and $m_i \in \{0, 1\}$, $i = 1, \ldots, n$.

After the voting process has finished the sum of votes will be decrypted multiplying by the private key d. Because of this the $(r_1 + \cdots + r_n) \cdot Q$ part will result in the point $[0 : 1 : 0]$ and $P_{m_1+\cdots+m_n}$ in the point R of the form:

$$R = [\rho : 1 : 0],$$

where ρ contains $(m_1 + \cdots + m_n) \cdot d \cdot N$.

From the point R we obtain R' by dividing the first coordinate of R by N:

$$R' = [(m_1 + \cdots + m_n) \cdot d : 1 : 0],$$

and the last step to get the final result is to multiply the first coordinate of R' by d^{-1} which is a multiplicative inverse of d in $\mathbb{Z}_N$:

$$m_{result} = (m_1 + \cdots + m_n) \cdot d \cdot d^{-1}.$$

To find the result with 1 and 0 knowing that there were n participants and knowing the final result m_{result}, we can obtain the number of voters for "accept" and "reject" as simply as: the m_{result} will be the votes "accept",

and "reject" are: $n - m_{result}$. When $m_{result} > n/2$ we can say that it was accepted, if not, it was rejected.

This was one of the possibilities for the referendum, but there is another one. If we encrypt the vote for "reject" as $-1 \pmod N$, so when this message is added to the total sum which contains the polling station, it will simply subtract 1 from the sum. So with this variation it is possible to include a vote "not sure/don't know" with a value 0 and vote for "accept" let it the same 1. When the voting process will go on, the result will be direct, as if its value is positive the vote for "accept" won, but if it is negative, the "reject" vote won. This makes the system get the final result faster, to know who won the voting process without making any mathematical operation, but it makes impossible to know the number of votes for "accept", "reject" and "neutral". So if it is only needed to know the final result, the fastest way is the second one, even having three possible responses, but if it is important to know the amount of votes for each category, the only way is the first one.

4.2 Multiple Candidates Voting

The referendum voting only gives us the possibility to have two or three possible responses, but what is about having four or more candidates. For this type, the main problem is how to identify each candidate from the other ones in a message. The answer is easy, we have information about the total number of candidates and the voters, so in relation to the maximum number of voters it is possible to assign a message for each candidate as a distance equal to the number of voters. For example, if we have n voters and k candidates, the message identifying each candidate can be distributed as it is shown on the following table:

candidate	c_1	c_2	c_3	$\ldots$	c_k
message	1	$n+1$	$(n+1)^2$	$\ldots$	$(n+1)^{k-1}$

When the election process is started, each voter will receive from the polling station the list of candidates and their respective messages, represented as a number. When the candidate is chosen, his respective message will be encrypted with Paillier cryptosystem and sent to the polling station where it will add the vote to the total sum. For example, for a voter i and his voting choice for a candidate m_i the encryption is creating a point over the elliptic curve:

$$V_i = r_i \cdot Q + P_{m_i},$$

where $P_{m_i} = [m_i \cdot N : 1 : 0]$ and $m_i \in \{1, n+1, (n+1)^2, \ldots, (n+1)^{k-1}\}$.

After being encrypted and sent to the polling station the vote V_i, we need to check the correctness of the vote with the MLS proofs and ensure that V_i holds a message in range $m_i \in \{1, n+1, (n+1)^2, \ldots, (n+1)^{k-1}\}$.

The server will collect the votes and add them to the total sum, obtaining:

$$\sum_{i=1}^{n} V_i.$$

The voting with multiple candidates has the same process as the referendum explained before, so we will not repeat all the same information and continue from the decryption of the sum of votes.

The last step is to multiply the first coordinate of R' by d^{-1} which is a multiplicative inverse of d in $\mathbb{Z}_N$:

$$m_{result} = (m_1 + \cdots + m_n) \cdot d \cdot d^{-1},$$

where $m_{result} = \sum_{j=1}^{k} x_j \cdot (n+1)^{j-1}$, being x_j = number of votes for the candidate c_j.

To obtain the result of the voting process, this case differs from the referendum. The decrypted number needs for a backpack treatment where it will be separated into a number of votes for each candidate. To separate this number we use some basic mathematical operations. First, we start with the candidate whose message number is the highest and divide the decrypted result by this number and the result will be the total number of votes for this candidate. For example, we have as a result number m_{result}, so first we calculate the number of votes for the last candidate c_k, who has the highest message number, denoted as φ_k and obtained as the quotient of the integer division:

$$\varphi_k = m_{result}/c_k.$$

After this operation we need to calculate the m'_{result} to calculate the results for the rest of candidates.

$$m'_{result} = m_{result} \pmod{c_k}.$$

For the others, the process is the same, the m'_{result} now take place as the total sum of votes and repeats the process until the first candidate c_1. When the process is finished it will have the total number of votes in favour of each candidate and then it is possible to publish the result.

Chapter 5

Details of the Implementation

In this project we decided to use two different programming languages: Sage and C++. This decision was made because of the great mathematical advantages of Sage and the ease of use object oriented programming with C++.

5.1 Sage implementation

Sage programming language is based on Python language and inherits most of the properties. Sage was made for working with mathematics and has some build-in libraries for working with elliptic curves, where the operations are optimized. At this point, creating an elliptic curve over p or q with Sage as simple as:

```
E = EllipticCurve(GF(p),[a,b])
E = EllipticCurve(GF(q),[a,b])
```

To get the cardinality of the created curve:

```
C = E.cardinality()
```

First of all, we implemented the original Paillier cryptosystem to see how it works. For this case the program had functions:

```
def key_generator(k):return landa, N, g;
def encryption(g,m,N):return c;
def decryption(c,landa,g,N):return message;
```

The first function $key\_generator(k)$ generates private and public values for working with Paillier cryptosystem of length k, which is an argument for the function. As a result, returns $landa, N, g$, where $landa$ is the private key of the cryptosystem, N is a modulo and g is a generator of the ring $\mathbb{Z}_N$.

The second one, $encryption(g, m, N)$ encrypts the message m with the public values g and N. The method returns the ciphertext of the message m.

The last one, $decryption(c, landa, g, N)$ is used to decrypt the cyphertext c passing the private key $landa$ and public information g and N. The returning value of this function is the original message.

With these functions declared, the last thing was to create a body which was going to test the correct generation of the keys, encryption and decryption of the cryptosystem. The following code was used for this task:

```
landa ,N, g = generation(10)
m=50
#Encr.
c=encryption(g,m,N)
print "Encrypted m=", c
#Decrypt
message=decryption(c,landa,g,N)
print "Decrypt m=",message
```

And the output of this test was:

```
Encrypted m= 97796126996
Decrypt m= 50
```

As it shows, the test was correctly executed and the original message is equal to the decrypted one.

After testing and proving that it works, the next step was to implement Elliptic Paillier cryptosystem. The functions used for this task were:

```
def encrypt(m,x,z,a,b,N):return S;
def decrypt(s1,s2,d,a,b,N):return message;
def add(x1,z1,x2,z2,a,b,N2):return x3, z3;
def double(x1,z1,a,b,N2):return x2, z2;
def mul(x1,z1,k,a,b,N2):return x2, z2;
def generate_point ( A , B , N2 ): return Q;
```

The function $encrypt(m, x, z, a, b, N)$ encrypts the message m with the public information a, b, N, and the coordinates x, z of the point P and returns the point S which is represented as an array of two positions, where $S[0]$ is the coordinate x and $S[1]$ coordinate z.

The next function, $decrypt(s1, s2, d, a, b, N)$ decrypts the point S with coordinates $s1$, $s2$ using private key d and public values a, b, N and returns an original message associated to the encrypted point.

The function $add(x1, z1, x2, z2, a, b, N2)$ makes the addition between two points P and R with coordinates $x1$, $z1$ for the first one and $x2$, $z2$ for the second one, using the public values a, b and $N2$. At the end, the function returns the coordinates $x3$, $z3$ of the new generated point $P + R$.

The next one $double(x1, z1, a, b, N2)$ doubles the point P with coordinates $x1$, $z1$ and the public information a, b, $N2$. The returning values are the coordinates $x2$ and $z2$ of the new generated point $P + P$.

The other function, $mul(x1, z1, k, a, b, N2)$ multiplies the point P with coordinates $x1$ and $z1$ by a scalar value k using public values a, b, $N2$. As a result, returns the coordinates $x2$, $z2$ of the new point $k \cdot P$.

The last one, probably the most problematic that was discovered while implementing the system. To find a point on the elliptic curve over $\mathbb{Z}_{N^2}$. It is quite difficult, as Sage does not offer any operation for getting this type of points over such a ring, so the function $generate\_point(A, B, N2)$ was designed. The arguments A and B are the public values of the generated elliptic curve over the ring $\mathbb{Z}_{N^2}$. As the result of the function, the returning values are coordinates x and z of the new generated point Q. The implemented algorithm is showed below:

```
while true :
      XXX = randint( 0 , N )
      XXX = Zmod( N )( XXX )
      ZZZ = randint( 0 , N )
      while gcd( ZZZ , N ) != 1 :
          ZZZ = randint( 0 , N )
      ZZZ = Zmod( N )( ZZZ )
      aux = (XXX^3+AAA*XXX*ZZZ^2+BBB*ZZZ^3)/ZZZ
      if is_square( aux ) :
          YYY = sqrt( aux )
          if gcd( ZZ( YYY ) , N ) == 1 :
              break
```

The next task was to create a code for testing the implemented system. The next code generates the keys of 64 bits:

```
T=64
p=random_prime(2^T)
q=random_prime(2^T)
N=p*q
a=randint(1,N)
b=randint(1,N)
if gcd(n,6*(4*a^3+27*b^2))!=1:
        print "error"
error=0
N2=N*N
G=IntegerModRing(N2)

Fn=Zmod(N)
```

```
Fp=Zmod(p)
Fq=Zmod(q)
Ep=EllipticCurve(Fp,[a,b])
Eq=EllipticCurve(Fq,[a,b])
CardP=Ep.cardinality()
CardQ=Eq.cardinality()
M=lcm(CardP,CardQ)

AAA = Zmod(N)(ZZ(a))
BBB = Zmod(N)(ZZ(b))
bool = false
while bool == false :
    x , z = generate_point( AAA , BBB , N )
    Q = mul(x,z,n,a,b,N2);
    aux = mul(Q[0],Q[1],M,a,b,N2)
    bool = aux != -1
print "Q IS:",Q
message = randint( 0 , n  )
print "Message: " , message
S2=encrypt( message ,Q[0],Q[1],a,b,N)
print "S=",S
result=decrypt(S[0],S[1],M,a,b,N);
print "The decryption=",result
```

The execution of this program gave the next output:

```
Q IS:
(629814227124152489689310215875548972790256611974717879209485257963094770 50,
509443144915051139447770324024107529139299922125493875149475679145683842385)
Message:  61542845502064658743406 1063128117019
S=
(462789889459845113177700878973838770131452856333304385881626047578778083692,
491386635717561656898096226579559219757326635301459927796708743182632673691)

The decryption=61542845502064658743406 1063128117019
```

The next case was to check the homomorphic property of the system and try that the addition of two encrypted messages will give the addition of their original messages after decryption. For this test the code for generating public and private values was not changed, so to make it shorter the changed part will be shown next:

```
message = 111
print "Message: " , message
S1=encrypt( message ,Q[0],Q[1],a,b,N)
```

```
S2=encrypt( message ,Q[0],Q[1],a,b,N)
S=add(S2[0],S2[1],S1[0],S1[1],a,b,N);
print "S=",S
result=decrypt(S[0],S[1],M,a,b,N);
print "The decryption=",result
```

The execution in this case gave the next output:

```
Q IS:
(441290227777683813175538504436087017707409582049138571729310015405171759668,
82389935639950073920524480198364643567549697842343229349454889915084055429 7)
Message:  111
S=
(457024503650383879497565607587423186019067713151801737191735962635258317737,
80423956400073738420230793828785067603456579730711866833415883375452797557 5)
The decryption=222
```

After implementing the algorithm, it was tested for different lengths of the prime numbers p and q. The tests showed that if we increase the length of the primes p and q, the time to calculate a point also grows. For example, to generate the point Q with the primes p and q of the length 64 bits, it was calculated in less than 5 seconds, for 128 bits the time increased up to $10 - 15$ minutes and 160 bits it was searching for 1-2 days. This is on one side, on the other, for implementing and simulating an electronic voting process Sage was not good. The idea was to implement the system using object oriented programming and Sage is not the best option for this, so we decided to choose another programming language such as C++.

5.2 C++ implementation

We used C++ language because it offers one of best performance, easy object oriented programming and a useful library called Crypto++ which contains all the tools needed for working with large integer numbers and modular arithmetic.

The first task was to design a class based model for electronic voting, as there will be voters, a server collecting votes and it is needed a trusted party which will store the private keys. This is because the server has not enough rights to know the private information.

The program was implemented with tree main classes:

$Voter$, $PollingStation$, $KSTP$ (trusted party).

A *main* function which creates objects of those classes and generates some type of electronic voting simulations. There is also a helping class which holds all needed operations with the points over the elliptic curves. Taking a close look at the implementation, $Voter$ class was defined as:

```
class Voter {
private:
    Integer n;
    Integer a;
    Integer b;
    Integer Qx,Qz;
    Integer Sx,Sz;
    Helper h;
public:
    Voter(Integer Qx,Integer Qz, Integer a, Integer b, Integer N);
//-------------------------------------------------------------
    void Vote(vector<Integer> &candidates, PollingStation &ps);
};
```

Out of this structure, it is possible to see that there is a constructor which receives as arguments the coordinates Qx and Qz of the point Q and a, b of the elliptic curve and the modulo N. This way, when the instance of this class was created, we save the public values inside. The other function is $Vote$ which receives the vector structure *candidates* containing the list of candidates and the other one is the instance of $PollingStation$ which receives the votes.

The next one is $PollingStation$ class with the next structure:

```
class PollingStation {
private:
    Integer n;
    Integer a;
    Integer b;
    Integer Qx,Qz;
    Integer Sx,Sz;
     Helper h;
     vector<Integer> candidates;

public:
    PollingStation();
//-------------------------------------------------------------
    void addVote(Integer x, Integer z);
//-------------------------------------------------------------
```

```
    vector<Integer> getResult();
//-------------------------------------------------------------
    vector<Integer> getCandidates();
//-------------------------------------------------------------
    Integer getA();
//-------------------------------------------------------------
    Integer getB();
//-------------------------------------------------------------
    Integer getN();
//-------------------------------------------------------------
    Integer getQx();
//-------------------------------------------------------------
    Integer getQz();
};
```

In this class the constructor does not receive anything, but inside initializes the public values. There are *getters* which return the public values. The function $addVote$ adds the point P with coordinates x and z to the accumulated sum of votes. The next one is $getResult$ which returns a vector structure containing the total number of votes for each candidate. The last one, $getCandidates$ which simply returns the list of candidates and their respective messages.

The $KSTP$ class which represents trusted authority, has the next structure:

```
class KSTP {
private:
    Integer n;
    Integer a;
    Integer b;
    Integer M;
    Helper h;
public:
    KSTP();
//-------------------------------------------------------------
Integer Decrypt(Integer Sx, Integer Sz);
};
```

This class has only one main job, to use *decrypt* function, to decrypt the point S with coordinates Sx and Sz and return an $Integer$ value which represents the decrypted message.

The last one is the $Helper$ class implemented this way:

```
class Helper {
public:
```

```
    Helper();
    //-------------------------------------------------------------
    string DecToBin(Integer number);
    //-------------------------------------------------------------
    void doble(Integer x1,Integer z1,Integer a1,Integer b1,
               Integer N,Integer& x3,Integer& z3);
    //-------------------------------------------------------------
    void add(Integer x1,Integer z1,Integer x2,Integer z2,Integer a1,
             Integer b,Integer N,Integer& x3,Integer& z3);
    //-------------------------------------------------------------
    void mul(Integer x1,Integer z1,Integer num,Integer a,
             Integer b,Integer N,Integer& x3,Integer& z3);
```

This one is a helping class to collect all the function which were repeating at each class. The main point was, if we needed to change anything at the functions, this change affects only this class, otherwise, we needed to check and change the implementation at each class. The first function is *DecToBin* which returns a binary representation of the integer *number*. The next one, *doble* doubles the point P with coordinates $x1$ and $z1$ and the public values a, b, N. We needed to return two values for this function, so we decided to store them at the pointers $x3$ and $z3$. The next one, *add* adds the point P with coordinates $x1$, $z1$ to the point R with coordinates $x2$, $z2$ and saves the result in $x3$ and $z3$. The last one is *mul* which multiplies the point P with coordinates $x1$ and $z1$ by a scalar *num* using public information a, b, N and saves the result at $x3$ and $z3$.

After implementing all the classes we created a *main* function which was simulating a voting process with different number of the candidates and voters. One of the tests have this *main* function:

```
PollingStation ps;
Integer times("200");
vector<Integer> can=ps.getCandidates();
Voter v(ps.getQx(),ps.getQz(),ps.getA(),ps.getB(),ps.getN());
cout<<"Voting has started!\n";
while(times>0){
v.Vote(can,ps);
times=times-1;
}
vector<Integer> res=ps.getResult();
for(int i=0;i<res.size();i++){
if (res[i]>res[winner])
 {winner = i;}
cout<<"CANDIDATE "<<res.size()-i<<"  ::"<<res[i]<<"\n";
}
cout<<"The winner of the elections is candidate:
```

```
      "<<res.size()-winner<<" with "<<res[winner]<<" votes.\n";
}
```

In this program, we create an object *PollingStation*, then define *times* which says the number of voters. After that, it gets and stores the vector with candidates, which is used for a better understanding of the implementation. Then, creates a $Voter$ object v, passing him public values from the *PollingStation* by using *getters*. The *while* loop is running *times* times and v object sends the vote using $Vote$ function to the object ps. When the loop has finished, the program will get the result from ps and find the winner. In the last step, prints the result.

The output for this execution was given in this form:

```
Voting has started!
CANDIDATE 4  ::56.
CANDIDATE 3  ::44.
CANDIDATE 2  ::50.
CANDIDATE 1  ::50.
The winner of the elections is candidate: 4 with 56. votes.
```

We can see that the execution was correct and the candidate number 4 got the highest amount of votes.

The reason of creating this type of structure was to separate different parts of electronic voting as much as possible. For example, voter has nothing to deal with the private key and the KSTP at any moment, as well as, PollingStation has no information on how to decrypt the sum of votes. On the other side, KSTP only decrypts and has no information about election process. With the implemented system we did the simulations with 100, 10000 and 10^6 voters sending random votes and proving that it gave a correct result.

Chapter 6

Results and Conclusions

In this chapter we are going to explain the results and conclusions of the finished work. Moreover, we present several proposals for the future work and improvements for the project.

6.1 Experimental Results

In this section we present several tables comparing the results of our implementation between the original Paillier cryptosystem and the elliptic Paillier cryptosystem.

On the first Figure 6.1 we compare the time needed to obtain the public information and private keys for both cryptosystems in function of key size.

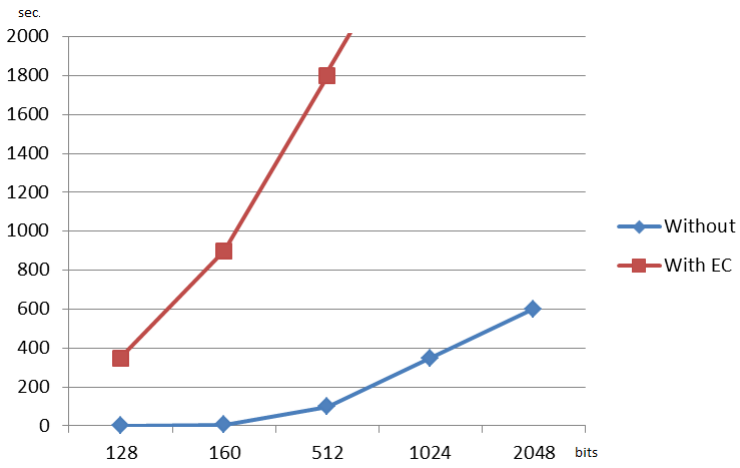

Figure 6.1: Time to generate initial information

From the Figure 6.1 it is easy to see that the time needed to find the values for elliptic curves is growing in an exponential way because of the

difficulty of finding a point over $\mathbb{Z}_{N^2}$ to work with. On the other side, the original Paillier cryptosystem, the time is growing in a linear way, so all the same information was obtained much faster.

In the next Figure 6.2 we compare the time which triggers voting process implemented with elliptic curves and with original in function of the number of voters.

We can see that the time grows in function of the number of voters. In both cases the systems are working with the same speed.

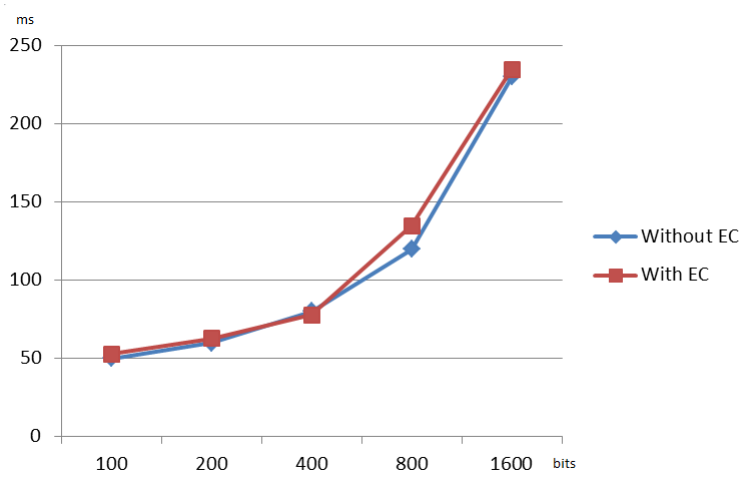

Figure 6.2: Time with number of voters

6.2 Conclusions

We have implemented an homomorphic electronic voting system using Paillier cryptosystem with elliptic curves. Since this cryptosystem has the homomorphic property, an advantage of this e-voting scheme is that we need just one decryption at the end of the voting process, corresponding to the sum of all the votes.

In cryptographic protocols based on the discrete logarithm problem the use of elliptic curves provide an important redaction of the key length, maintaining the same level of security compared to the classical protocols. Nevertheless, regarding our proposal we have realized that elliptic Paillier cryptosystem does not help us to reduce the amount of bits required for security. This is due to the fact that Paillier cryptosystem bases its security on the integer factorization problem and composite residuosity problem. Thus, in order to guarantee a suitable level of security we need to use primes p and q of 1024 bits to generate the keys. Because of this, the initialization of pub-

lic and private information for Paillier cryptosystem using elliptic curves is slower, compared to its original version.

This way, we can say that elliptic Paillier cryptosystem is not, currently, the best option for the electronic voting processes. That is why we leave this project open for future work and improvements.

6.3 Future Work

After finishing the project we can define some options for future work:

- Make improvements for the elliptic Paillier cryptosystem, to hide some public values and make it more secure.
- Implement any faster way to find the point over the curve $\mathbb{Z}_{N^2}$.
- Implement MLS tests for the system.
- Compare our protocol with other homomorphic tallying proposals.

Moreover, the generated code can be used for the further work and for implementing new systems.

Bibliography

[1] M.A. Cerveró, V. Mateu, J.M. Miret, F. Sebé, and J. Valera. An elliptic curve based homomorphic remote voting system. *RECSI 2014*, 2014.

[2] D. Chaum. Blind signatures for untraceable payments. *Advances in Cryptology: Proceedings of Crypto 82*, pages 199–203, 1983.

[3] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE trans. Inform. Theory*, pages 469–472.

[4] S. Galbraith. Elliptic Curve Paillier Schemes. *Journal of Cryptology 15*, pages 129–138, 2002.

[5] X. Lai. On the design and security of block ciphers. Phd thesis, ETH Zurich, 1992.

[6] V. Mateu. Implementació d'un Sistema de votació Sobre la xifra de Paillier i Elgamal. Treball Final de Master, Universidad de Lleida, 2009.

[7] V. Mateu. Votació electrònica amb recompte homomòrphic. Treball Final de Carrera, Universidad de Lleida, 2010.

[8] V. Mateu, J.M. Miret, and F. Sebé. Verifiable encrypted redundancy for mix-type remote electronic voting, LNCS 6866. *EGOVIS 2011*, pages 370–385, 2011.

[9] NIST (National Institute of Standarts and Technology). AES (Advanced Encryption Standart). *Federal Information Processing Standart (FIPS)*, 2001.

[10] NIST (National Institute of Standarts and Technology). Recommendation for transitioning the use of cryptographic algorithms and key lengths. *Federal Information Processing Standart (FIPS)*, 2011.

[11] US National Bureau of Standarts. DES (Data Encryption Standart). *Federal Information Processing Standart (FIPS)*, 1977.

[12] P. Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. *Procs of EUROCRYPT'99*, pages 223–238, 1999.

[13] R. Rivest. The MD5 Message-Digest Algorithm. *MIT LCS and RSA Data Security*, 1992.

[14] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM 21, no 2*, pages 120–126, 1978.