EFFICIENT CRYPTOSYSTEM FOR UNIVERSALLY VERIFIABLE MIXNETS

Sandra Guasch Castelló

Company Supervisor: Jordi Puiggalí Allepuz
Academic Supervisor: Miquel Soriano Ibáñez

Bachelor Degree of Telecomunication Engineering

Universitat Politècnica de Catalunya, ETSETB
Scytl Secure Electronic Voting
December 2009

This PFC has been done by means of a university-company collaboration program in Scytl Secure Electronic Voting.

Agraïments

M’agradaria primer de tot agrair an es meus pares i an es meu germà es recolzament que m’han donat aquests últims dies de projecte, tot i que no acabéssin d’entendre sa feina i presió que comporta. Sobretot a sa meua mare, que m’ha fet correccions d’anglès a sa introducció i a ses conclusions “perquè lo altre, que és més tècnic, ja ho sabràs tu millor”.

Després d’ells, es següent que és mereix es meu agraïment és en Ferran, que ha tengut (i té) infinita paciència amb es meus histerismes, dubtes i queixes. Amb ell he compartit s’últim any de sa carrera i es procés de realització des projecte, i poder comptar sempre amb una persona que estava en sa mateixa situació que jo m’ha ajudat molt. Grasis!

També m’agradaria agrair as meus companys de classe de sa enginyeria superior es anys de carrera, durs però entretenguts, i a ses meues amigues de sa tècnica i d’Eivissa poder-nos ajuntar de quan en quan per fer uns cafès, contar-nos ses nostres experiències en es camins diferents que hem près, i riure’ns un poc de tot.

No puc deixar d’anomenar aquí a la Paz Morillo, que ha estat sempre allí per a qualsevol dubte que he tengut.

Finalment, m’agradaria agraïr al Miquel Soriano i al Jordi Puggalí sa oportunitat que m’han donat de entrar a treballar a Scytl per fer allí es meu PFC, i an es meus companys de feina, sobretot a en Miquel Comas i a en Jaume Poch, per fer de sa feina un lloc més agradable i entretengut.

Index of Contents

Index of Figures

1. Introduction

1.1. Democracy and electoral processes:

Experts say that the exercise of democracy is not reduced to electoral practices. Nonetheless, modern democracy cannot be conceived without elections, in such a way that the fundamental indicator of the democratic societies is the conduction of free elections. In many countries they are, for the majority of the citizens, the privileged way to be in touch with politics.

Lots of interests come into play at the time of determining the government successors, and numerous fraud cases have been detected in electoral processes: disappeared or found ballot boxes, manipulated counting machines, voter coercion, vote buying, ballot manipulation, etc. Since the elections are a critical process in a democratic society, the legitimacy of the results must be ensured.

The fast technologic growth gives every time more tools to the society in order to enhance their quality of life and make easier the daily tasks, then it is natural for the electoral processes to evolve technologically too.

There are several reasons to use e-Voting systems instead of conventional mechanisms, for example a more comfortable election management for especially complex processes, a faster ballot counting to obtain earlier results, or a cheaper election process.

The use of remote media as mobile phones or Internet in order to vote has other advantages:

One of the main objectives of using new technologies in the democratic processes is to avoid the low election turnout that characterizes an electoral process nowadays, especially among the youngest sectors. The use of remote applications can make the electoral process more appealing to the youngest voters that, after all, use the telecommunication systems in their daily life.

Another advantage of remote voting is that it can make easier the electoral process to people with reduced mobility or with any disability. These persons can find a lot of impediments when they go to vote in an Electoral College, while if they can vote in a remote way they can find suitable places to do it, or use specific equipments they already have at their homes. Moreover, electronic voting can provide techniques to increase the system accessibility and usability.

In the same way, the voters abroad can find in remote voting an opportunity to express their willing in a more comfortable, convenient and reliable way than with postal voting.

In order to achieve a security level in remote e-Voting at least as high as in a conventional election, or even higher, cryptographic applications must be used.

1.2. Security in remote e-Voting

Security is of paramount importance in any voting process. The introduction of e-Voting makes the process less transparent. Therefore, additional security requirements must be considered.

Remote e-Voting makes use of cryptography in order to provide a secure service, as it is required in an electoral process.

Voters must be able to verify that their voting intent has been properly recorded in the voting system (cast as intended) and counted in the vote tally (counted as cast) at the same time that coercion and vote buying must be prevented. These requirements create a trade-off between verifiability and vote secrecy that the electronic voting system has to solve in an efficient way.

There are certain security requirements that an electoral process conducted electronically must comply with:

- Voter authentication: it must be ensured that each vote is cast by an eligible voter and that just one vote per voter is tallied.

- Voter privacy: while it must be ensured that a just the eligible voters can cast a ballot, it must be impossible to connect the voter identity with the content of his/her cast vote.

- Accuracy of the Election Results: it must be impossible for anyone to erase or modify votes submitted by eligible voters, or to add invalid votes on behalf of abstaining voters.

- Intermediate result privacy: the election intermediate results must be secret until the election process has ended in order to not to influence the voters that have not yet participated in the voting process.

- Vote verifiability: voters must be able to verify independently that their votes have been included in the final tally and counted properly.

- No coercion: it must be impossible for a voter to demonstrate the content of his/her cast vote in order to prevent coercion or vote buying.

One of the above properties that are more enforced in an e-Voting system and that is more present in conventional elections too is the voter privacy. In turn, it is a complex requirement since, while the voter privacy must be preserved, it is necessary to ensure that the voter who casts the vote is an eligible one. The design of a system that complies efficiently with these two requirements is a real challenge. An efficient way to solve it is using advanced cryptographic protocols.

1.3. Securing elections using basic cryptography

In a basic implementation, the e-Voting systems use standard cryptographic techniques to ensure the voter privacy and the election reliability. The most common cryptographic tools to be used in e-Voting processes are the vote encryption and its digital signature.

1.3.1. Vote encryption

A vote can be encrypted using two types of cryptographic algorithms:

- Public key algorithms (asymmetric): the election has two keys, a public one that is known by all the voters and a private one that is known by just one electoral entity (usually the Electoral Board). The voters use the election public key in order to encrypt their choices and, at the time of the tally, they are decrypted using the election private key. That way it is ensured that anybody but the electoral entity, that has the private key, can access the votes' content.

  Being $\mathrm{eb_s}$ the election private key, $\mathrm{eb_p}$ the public one and $\mathrm{E, D}$ encryption and decryption operations, an electronic ballot v is encrypted by a voter as

  $$\mathrm{E_{eb_p}(v)}$$

  In the decryption process the encrypted electronic ballot is decrypted by the electoral entity as

  $$\mathrm{v = D_{eb_s}(E_{eb_p}(v))}$$

- Secret key algorithms (symmetric): in this case the voter and the electoral entity share a unique secret key used to both encrypt and decrypt the votes. In order to prevent a voter to be able to decrypt the other voters' choices, a unique secret key is generated for each voter, while the electoral entity has access to all the keys.

  Being $\mathrm{k_i}$ the symmetric election key shared by a voter i and the election entity, an electronic ballot v is encrypted by a voter as

  $$\mathrm{E_{k_i}(v)}$$

  In the decryption process the encrypted electronic ballot is decrypted by the electoral entity as

  $$\mathrm{v = D_{k_i}(E_{k_i}(v))}$$

Compared with asymmetric algorithms, the symmetric algorithms have scalability problems due to that a different key has to be stored for each voter in the voting system. In the other hand, the asymmetric algorithms are slower in the decryption process than the symmetric ones, especially for large messages.

In order to solve these problems it is usual to use hybrid systems that combine the advantages of symmetric and asymmetric algorithms.

In a hybrid encryption algorithm the message is encrypted using a symmetric key generated by the voter at random at the time of encryption. Then, this symmetric key is encrypted using the election public key. This way just two keys are needed for the electoral process (public and private), due to that with the private key the electoral entity can access the symmetric keys used to encrypt the votes (faster than decrypting the entire ballot since the symmetric key is supposed to be shorter) and decrypt them faster than using an asymmetric cryptosystem. This encryption system is known as digital envelope.

Being eb_s, eb_p the election private and public keys and k_i a random session key generated by the voter at the moment of the encryption, an electronic ballot v is encrypted as

$$\left[E_{k_i}(v), E_{eb_p}(k_i)\right]$$

In the decryption process the encrypted electronic ballot is decrypted by the electoral entity as

$$k_i = D_{eb_s}\left(E_{eb_p}(k_i)\right)$$
$$v = D_{k_i}\left(E_{k_i}(v)\right)$$

1.3.2. Digital signature

A vote digital signature is performed using an asymmetric key algorithm. Usually the voter calculates a Hash operation over his/her encrypted vote and then “encrypts” the Hash result using his/her private key (instead of using the public key as in the encryption process). The voter public key is known by everybody; therefore anyone can verify the digital signature “decrypting” the result of the Hash function with the voter public key and comparing that the Hash of the signed encrypted vote matches the “decrypted” Hash. The digital signature is performed over the encrypted ballot. This prevents the correlation between the decrypted vote and the digital signature, since signatures are only connected to encrypted votes.

Being k_{is}, k_{ip} the voter i private and public keys and *H* a Hash function, an electronic encrypted ballot v is signed as

$$\left[E(v), E_{k_{is}}\left(H\left(E(v)\right)\right)\right]$$

The electoral entity verifies the signature on the encrypted ballot as

$$x = D_{k_{ip}}\left(E_{k_{is}}\left(H\left(E(v)\right)\right)\right)$$
$$\text{Verify that } x = H\left(E(v)\right)$$

Digitally signing the encrypted votes allows the verification of the vote integrity once it is received in the voting server, due that if it is modified after the submission step the Hash function will not match the one encrypted with the voter private key. It also allows the verification of the voter eligibility, due that the voter has a digital certificate that connects his/her identity with the public key. This method is similar to the one in postal voting where the voter sends the ballot inside of a signed envelope and the signature is checked at reception time. Once the ballot is extracted from the envelope it cannot be connected to the submitter identity.

The scheme for this election model is the one in figure 1.1:

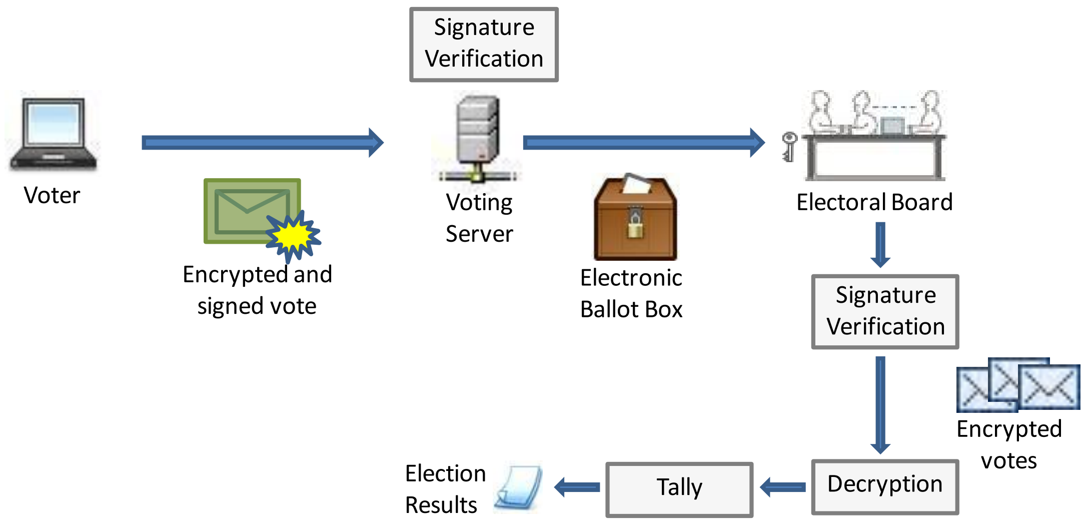


Fig. 1 - 1 Electronic voting using basic cryptography

Although this scheme seems secure enough to protect the vote secrecy, the voter privacy can be broken if someone tracks the order the signed and encrypted votes are received and the order they are decrypted, correlating the ballot content for an identified voter. Moreover, this system does not allow the verification that the ballots are not manipulated between the vote submission and the decryption steps.

Therefore, we need more advanced cryptographic methods to comply with the security requirements.

The main objective of this PFC is to analyze some advanced cryptographic methods and design a cryptosystem that could be used in e-Voting environments in order to encrypt the votes in an efficient way, making it suitable to be used with the advanced cryptographic methods that have been studied.

2. e-Voting protocols focused in the protection of the voter privacy

There are various methods in electronic voting processes designed to protect the voter privacy implementing voter anonymization systems. These methods can be classified depending on the time at the voting process when they work:

- Pre-election phase: it is the phase where the election configuration processes are done, as generating the voting ballots, closing the electoral rolls or assigning credentials to the voters (i.e. digital certificates).

  The methods that implement cryptographic processes in order to protect voter privacy in this phase apply cryptographic measures in the design of the voting ballots.

- Voting phase: this phase covers the period when the voters can cast their votes. During this phase the voters access to the voting system, identifying themselves and submitting their votes.

  The protocols that work in this phase apply the cryptographic measures after the voter identification and before she submits her vote, achieving an anonymous channel for the voter whereby the vote is separated from her identity.

- Counting phase: this phase starts when the voting period finishes. During this phase the digital ballot boxes are collected, and a vote tally is done in order to obtain the election result.

  The cryptographic protocols used to achieve the voter anonymity in the counting phase are focused on breaking the correlation between encrypted votes and decrypted contents.

In the next sections these systems are described in detail.

2.1. Protocols that implement anonymity in the pre-election phase

As mentioned before, in these protocols the methods to achieve the voter privacy are implemented in the ballot design phase.

These methods [1], [2], [52] are known generally as *Pollsterless*, since they do not need a device with enough computation capacity to encrypt the voter chosen options (since they are pre-encrypted). An advantage of these methods is that they can be used to vote by mobile phones using SMS text messages.

In this design phase different codes for the voting options are generated for each ballot. Once they have the codified voting options, the ballots are assigned randomly to the voters, and the parameters used to decode them are stored in a secure way in order to decrypt the votes during the tally.

The ballot design mechanism is simple:

- A random unique identifier is assigned to each ballot *i*:

  $$Id_i = Rand(), 1 \leq i \leq n,$$

  where *n* is the number of paper ballots.

- For each party or candidate in the election, C_j, a code $CC_{i,j}$ is generated based on the unique identifier of the ballot Id_i and the secret parameter *Kc*. Usually a unidirectional function like a MAC is used to calculate the codes:

  $$CC_{i,j} = MAC\left(MAC\left(C_j, Id_i\right), Kc\right), 1 \leq i \leq n, 1 \leq j \leq k,$$

  where *k* is the number of candidates in the election.

- For each candidate code $CC_{i,j}$, a return code $CR_{i,j}$ is generated using the secret parameter *Kr* and a non-reversible operation:

  $$CR_{i,j} = MAC\left(MAC\left(CC_{i,j}, Id_i\right), Kr\right), 1 \leq i \leq n, 1 \leq j \leq k$$

  Thus, for each candidate and for each ballot a different code and its return code are generated.

- Before starting the voting process, the ballots are sent by post in a random way to the voters. The paper ballots received by the voters are like this:

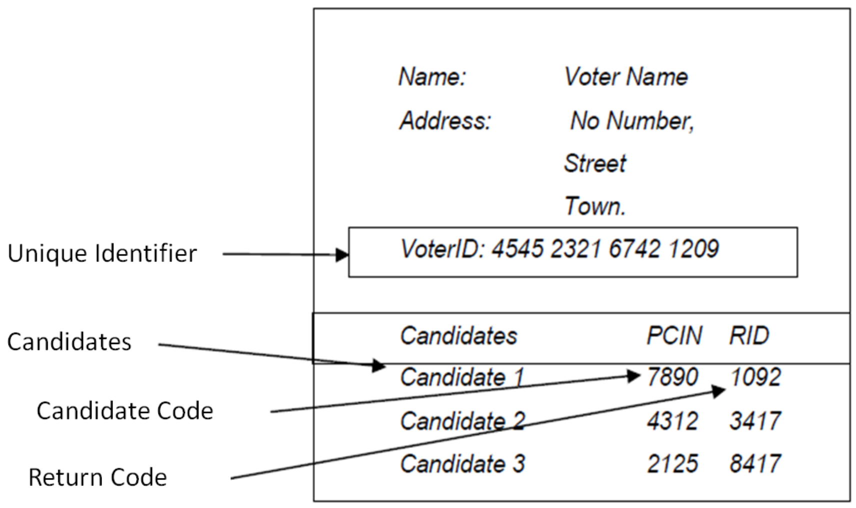


Fig. 2 - 1 Example of paper ballot in Pollsterless

- Assuming the voter uses a web interface, once the voting process has started, the voter accesses a web application where she is requested to insert the codes $CC_{i,j}$ belonging to the chosen candidates and the ballot identifier Id_i. Then, the voting server calculates the return codes $CR_{i,j}$ using the received candidate codes and the secret *Kr*

  $$CR_{i,j} = MAC\big(MAC\big(CC_{i,j}, Id_i\big), Kr\big)$$

  and sends them to the voter, that can compare if their value is the same that those assigned to the candidates in her ballot.

That way, the voter can verify that the vote received in the voting server contains the same options that she has chosen. An attacker is not able to know in advance the return code that must return in case of vote interception.

Moreover, in case that someone intercepts the codes, it is infeasible to know for which candidates the voter has voted for without knowing the contents of the paper ballot, so the voter privacy is preserved.

- Finally, the candidate codes are calculated again in the counting phase using the secret parameter *Kc*, in order to connect the received codes with the candidates:

  $$CC_{i,j} = MAC\big(MAC\big(C_j, Id_i\big), Kc\big)$$

Due that the system security is mainly based in the paper ballots, there are several methods to prevent their contents to be intercepted or manipulated before they arrive to the voters, such as printing the candidate codes and the return codes separately, or dividing the ballots and sending their pieces using different channels.

This system has several drawbacks:

- Usability: the voter has to input large character chains that compose the ballot identifier and the candidate codes, and it can be easy to make mistakes. Moreover, once the voter has voted, she has to verify the received return codes comparing them to the ones in the paper ballot next to her choices, which can be specially difficult for old people.

- Voter coercion or vote buying/selling could be possible in case that the received candidate codes were published and the voter showed the paper ballot with her codes to the coercer or the buyer, in order to verify that those connected to the desired candidates are published.

- Privacy could be compromised: an attacker could know the vote intention of a voter if he/she had access to the codes in the ballot.

- In case of interception an attacker could swap the codes in order to convince the voter that she has chosen a determinate candidate, while she is really choosing another.

The voter anonymity is preserved since the submitted codes are not digitally signed.

2.2. Protocols that implement anonymity in the voting phase

The aim of these methods is to achieve an anonymous channel for the voter, in such a way that the submitted vote is not connected to her identity.

This protocol, first presented in [7] and then enhanced in [3], is usually called the two agencies model, due to that it is based on two different services:

- The Validator Service identifies the voter, checks that she is in the electoral roll (her eligibility) and allows her to vote in an anonymous way.

- The Vote Service receives anonymous votes that have been validated by the Validator Service.

Usually, the blind signature protocol is used in these schemes. This protocol digitally signs an encrypted vote without having knowledge of its content. Thus, the privacy of the voter is preserved. The blind signature mechanism is based in the mathematical properties of an encryption system, such as the RSA:

Supposing that the entity that signs the encrypted votes has a public RSA key *(e, n)* and a private one *d*, this protocol implements the following steps:

1. The requestor (the voter) chooses a number *r* such that

   $$gcd(r, n)=1$$

2. Then, she generates a blinding factor *b* as

   $$b = r^e$$

3. The message to be signed is multiplied by the blinding factor

   $$m' = m \cdot r^e \; mod \; n$$

4. Finally, the requestor sends the blinded message *m'* the signer entity, that returns the digital signature of m' (*s'*) to the requestor, where

   $$s' = m'^d \; mod \; n$$

5. The requestor multiplies the signed blinded message by the inverse of the blinding factor and obtains the original message *m* signed by the signer entity

   $$s = s' \cdot r^{-1} \; mod \; n = (m \cdot r^e)^d \cdot r^{-1} \; mod \; n = m^d \cdot r \cdot r^{-1} \; mod \; n = m^d$$

Thus, in this protocol, the Validator Service receives a blinded encrypted vote jointly with voter identity, it checks that she is an eligible voter and signs the blinded encrypted vote. Then, the voter unblinds the vote and sends the recovered encrypted and signed vote to the Voting Service, which stores only the encrypted votes signed by the Validator.

The voting phases are the following:

- The voter creates a message that contains her chosen options, encrypts, blinds and sends it to the Validator Service.
- The Validator Service receives the encrypted message, verifies that the voter is eligible and digitally signs the blinded message, sending it back to the voter.
- The voter receives the blinded signed encrypted message and removes the blinding factor in order to obtain the original encrypted message, still signed by the Validator Service. Then, the voter sends it to the Voting Service.
- The Voting Service verifies that the encrypted message has been signed by the Validator Service and stores it.

A scheme of the communication process is shown in figure 2.2:

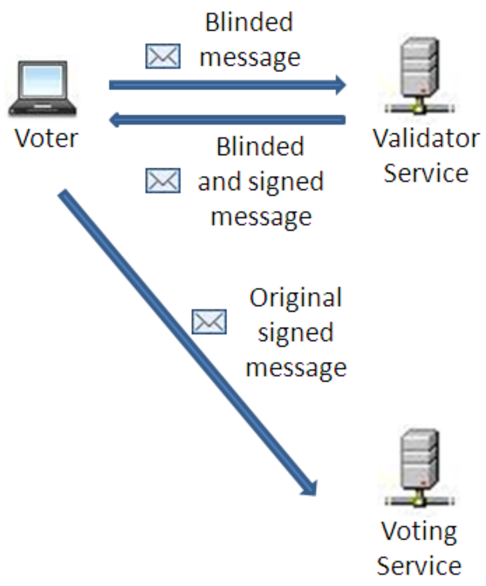


Fig. 2 - 2 Communications in a two agencies model

The voter anonymity is preserved since:

- The vote is blinded and then signed by the Validator Service.
- The voter has not to be authenticated to submit the vote to the Voting Service.
- The voting options are encrypted before they are sent.

Although this type of protocol is very efficient in order to preserve the voter anonymity when casting the vote, they have some drawbacks that prevent their implantation in real election processes:

- Since the Voting Service stores all the votes signed by the Validator Service, an attacker controlling the Validator Service could generate an unlimited set of non-valid votes that would be accepted by the Voting Service, modifying the election results. Just one entity compromised (Validator Service) results in a possible electoral fraud.

- The communications between the voter, the Validator Service and the Voting Service could be traced in order to relate the vote content with the voter identity, i. e. tracking the IP directions or using cookies.

- In case that the Validator Service and the Voting Service cooperated sharing information, they could correlate votes with voter identities.

2.3. Protocols that implement anonymity in the counting phase

These methods can be divided in two types:

- Systems that calculate the final tally without decrypting the votes individually, in such a way that the cleartext vote is never recovered and therefore, it cannot be correlated to the voter. These are called homomorphic tally protocols.

- Mixnets: these systems break the correlation between the reception order of the encrypted votes and the order in which the votes are decrypted.

Both systems operate in an election characterized by:

- Voters encrypt the vote before submitting it through the net. The encrypted vote contains the different options chosen by the voter.

- Once the voter has encrypted the vote, she also digitally signs it in order to verify its integrity and the voter eligibility when it is received in the Voting Service. In order to sign the vote to be cast, the voters must have a key pair belonging to a public key cryptosystem (usually this involves management of digital certificates).

- There is an entity, the Electoral Board, that is composed of people with different interests (i.e. a representative of each party) in order to prevent collusion. The Electoral Board has a key pair belonging to a public key cryptosystem: the public key is the one that the voters use to encrypt their votes, while the private key, only known by the Electoral Board members, is the one that can recover the content of the encrypted votes.

2.3.1. Homomorphic tally protocols

These protocols, introduced in [4], make use of the homomorphic properties of some cryptosystems whereby certain operations over the encrypted votes result in the encryption of the operation of the cleartext votes.

Being two votes v_1 and v_2, an encryption operation E and two algebraic operations Φ and Θ, the homomorphic property can be defined as

$$E(v_1)\ \Phi\ E(v_2) = E(v_1\ \Theta\ v_2)$$

There are some types of homomorphism depending on the definition of the operation Θ: the additive homomorphism and the multiplicative one:

- In a cryptosystem with additive homomorphic properties the multiplication of the encrypted votes results in the encryption of the addition of the cleartext votes. Thus, when this result is decrypted, the sum of the votes is obtained.

- In a cryptosystem with multiplicative homomorphic properties the multiplication of the encrypted votes results in the encryption of the multiplication of the cleartext votes. When this result is decrypted, the product of the votes is obtained.

Depending on the type of homomorphism that we want to use, we have to choose a cryptosystem with the appropriate homomorphic properties.

2.3.1.1. Additive homomorphism

When an additive homomorphic cryptosystem is used, the encrypted votes are usually formed by an array of parameters, each one corresponding to a certain candidate depending on the position in the array. Each parameter is composed by a fixed number exponentiated to a coefficient *a* if the candidate that represents is chosen or a coefficient *b* if not. One possible configuration is that the coefficients *a* and *b* are equal to *1* and *0.*

A common configuration scheme is presented in figure 2.3:

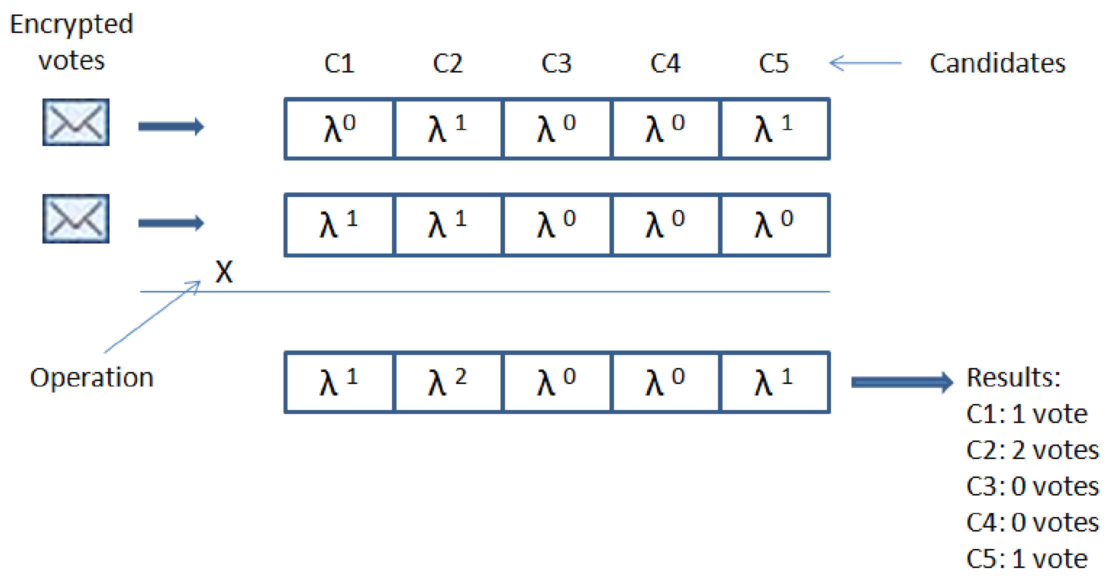


Fig. 2 - 3 Example of an additive homomorphic tally

Since the multiplication of the encrypted votes results in the addition of the exponents in λ, when the result is decrypted the addition of votes for each candidate is obtained.

Another possibility to encrypt the options is to have the parameters *a* and *b* equal to *1* and *-1.* Then, the result of the exponent addition is not equal to the votes for each candidate, but to the difference between positive and negative votes. Following the procedure described in [5], the results for each candidate can be obtained.

Some cryptosystems with additive homomorphic properties are the exponential ElGamal or Paillier.

For example, in exponential ElGamal a vote in an election where there are three candidates can be encrypted as

$$(g^{r_1} \, mod \, p, \lambda^{\alpha} \cdot h^{r_1} \, mod \, p),(g^{r_2} \, mod \, p, \lambda^{\alpha} \cdot h^{r_2} \, mod \, p),(g^{r_3} \, mod \, p, \lambda^{\alpha} \cdot h^{r_3} \, mod \, p),$$

where $(g, h, \lambda, \mathrm{p})$ are public parameters, r_1, r_2, r_3 are random integers, and $\alpha = \{0,1\}$ (details of ElGamal encryption can be found at Section 3).

If a voter selects the second candidate (the modular operations are omitted from now)

$$(g^{r_1}, \lambda^0 \cdot h^{r_1}),(g^{r_2}, \lambda^1 \cdot h^{r_2}),(g^{r_3}, \lambda^0 \cdot h^{r_3})$$

The multiplication of *n* votes results in

$$(g^{\sum_{i=1}^{n} r_{1i}}, \lambda^{\sum_{i=1}^{n} \alpha_i} \cdot h^{\sum_{i=1}^{n} r_{1i}}),(g^{\sum_{i=1}^{n} r_{2i}}, \lambda^{\sum_{i=1}^{n} \alpha_i} \cdot h^{\sum_{i=1}^{n} r_{2i}}),(g^{\sum_{i=1}^{n} r_{3i}}, \lambda^{\sum_{i=1}^{n} \alpha_i} \cdot h^{\sum_{i=1}^{n} r_{3i}})$$

After the decryption process we obtain $\lambda^{\sum_{i=1}^{n} \alpha_i}$ for each candidate and the discrete logarithm must be broken in order to obtain $\sum_{i=1}^{n} \alpha_i$ for each one. The break of the DL is based on the fact that the exponent is small and in trapdoor functions (in Paillier). Therefore in a large election the votes can be tallied dividing them in groups.

2.3.1.2. *Multiplicative homomorphism*

When a cryptosystem with multiplicative homomorphic properties is used, it is usual for each voter just to encrypt the selected options. Then, to perform the homomorphic tally, the encrypted votes are multiplied. The result of the decryption is the multiplication of all the candidates selected by the voters. In order to extract the votes for each candidate from this result, it is usual to use prime numbers to identify each candidate and extract the selected options using a factorization algorithm.

A scheme of the system is shown in figure 2.4:

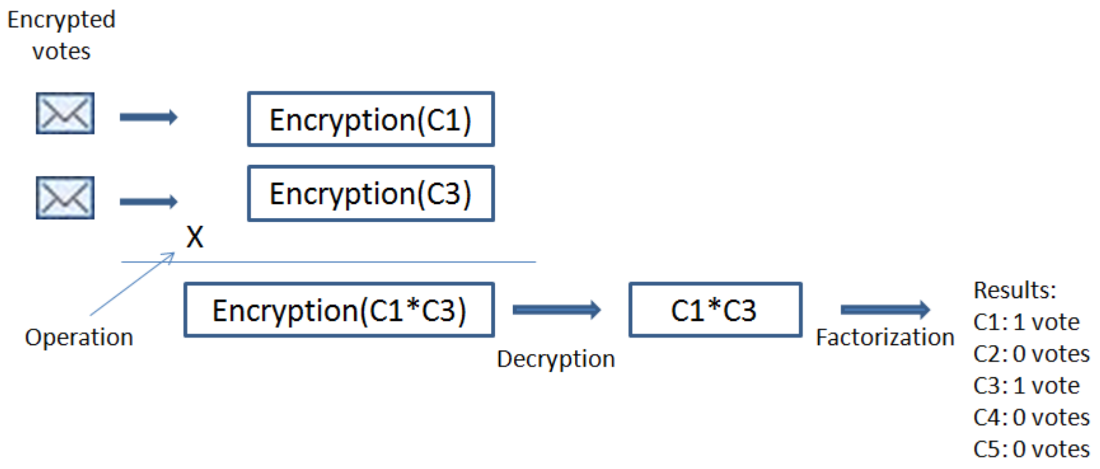


Fig. 2 - 4 Example of a multiplicative homomorphic tally

ElGamal, or RSA are cryptosystems with multiplicative homomorphic properties.

For example, when the ElGamal cryptosystem is used in an election with three candidates where each vote can choose just one of them, an encrypted vote is

$$(g^r, \mathrm{c_j} \cdot h^r),$$

where (g, h) are public parameters, r is a random integer, and $\mathrm{c_j}$ is the code for the chosen candidate.

The multiplication of *n* votes results in

$$(g^{\sum_{i=1}^{n} r_i}, \prod_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{c_i} \cdot h^{\sum_{i=1}^{n} r_i})$$

After the decryption process we obtain the product of all the candidates $\prod_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{c_i}$. Then, we can extract the results for each candidate, for example factorizing the result, and obtain $\mathrm{c_1}^{\alpha_1}, \mathrm{c_2}^{\alpha_2}, \mathrm{c_3}^{\alpha_3}$, where $\{\alpha_1, \alpha_2, \alpha_3\}$ are the total number of votes for each candidate.

2.3.1.3. Benefits and drawbacks

The homomorphic tally has some advantages in front of other systems.

- For example, due that the individual votes are never decrypted, it is impossible to break the voter privacy because the cleartext votes do never exist to be related to their authors.

- Moreover, a unique decryption process (or few of them) after the operation of all the encrypted votes results in a fast counting system.

- Since the decryption process is done once, it could be performed in a distributed way, using a secret sharing scheme where the private key to decrypt the votes is split, and each share is owned by one different party that partially decrypts the result, increasing the system security (i.e. using multi-party computation techniques). If each vote had to be decrypted individually, this scheme would not be efficient.

However, there are some drawbacks that difficult the implementation of these methods in common elections:

- Since the votes are never decrypted individually, it is necessary to ensure that the encrypted votes are well-formed. Otherwise the results could be falsified (i.e. voting several times for a candidate in a single vote). The verification process needed to ensure that the votes are well-formed requires complex Zero Knowledge Proofs that imply high computation costs for the voter application that creates the proofs and for the voting system that verifies them. Some of these proofs are explained in Section 5.

- The hybrid encryption algorithms, such as digital envelopes, cannot be used in these systems, due that the votes have to be encrypted using an homomorphic asymmetric cryptosystem. Then, the properties of the symmetric cryptosystems cannot be used and the encryption and decryption processes are slower than if we could use them.

- The vote format must support the operations of the individual voting options and therefore, it is more restrictive than other schemes. For example, when additive homomorphic cryptosystems are used, the encrypted votes are formed by an array of parameters composed by a fixed number exponentiated with a certain number depending on the selections. Thus, all the possible voting options must be encrypted. If the election is complex and has a medium or large amount of candidates, this results in large and difficult to manage encrypted votes. When multiplicative homomorphic cryptosystems are used, suitable prime codes have to be selected to represent the candidates, and an efficient factorization algorithm has to be used.

- Since all the possible voting options have to be prefixed, a voting system that uses homomorphic tally does not allow the use of open questions or texts written by the voters (commonly known as write-ins).

As result of those drawbacks these systems are commonly used in small and simple elections. Otherwise the process complexity increases too much.

2.3.1.4. Additive versus multiplicative homomorphism

As part of the project, we compared both technologies. Although the additive homomorphic systems are more used (since they provide in a direct way the total number of votes for each candidate) in electronic voting that the multiplicative homomorphic ones, they are more inefficient for several reasons [6]:

- If Paillier encryption is employed (additive homomorphism), the following drawbacks in efficiency exist:

    - <u>Inefficient set-up:</u>
      In voting schemes, the private key of the encryption algorithm is usually generated and shared by multiple trustees, so that there is no need to trust any single party to achieve vote privacy. As the private key is a factorization secret in Paillier encryption, distributed key generation is highly inefficient. In

comparison, distributed key generation in ElGamal (distributed generation of a secret logarithm as the private key) is more efficient.

  - Multiple encryption:
    Usually, a voter has to perform an encryption for each candidate and prove that each of her encryptions contains a valid message.

  - Inefficiency of multiplicative and exponentiation computations:
    In Paillier encryption, each multiplication is performed modulo n^2, where n is the product of two large primes. In comparison, in original ElGamal encryption, each multiplication is performed modulo p, a large prime. If the same security strength is required, n and p should have the same length (i.e. 1024 bits). Paillier indicated that a multiplication in a Paillier encryption scheme is more than three times as costly as a multiplication in an Elgamal encryption scheme when n and p have the same length.

- If the exponential ElGamal encryption (additive homomorphism) is employed, the following drawbacks in efficiency exist:

  - Multiple encryption:
    Usually, a voter has to perform an encryption for each candidate and prove each of her encryptions contains a valid message.

  - Inefficient Discrete Logarithm search:
    As stated before, a search for logarithm is needed in the decryption function. Even though the (currently known) most efficient solution for DL in a certain interval — Pollard's Lambda Method — is employed, $0.5\log_2 n$ exponentiations, $O(n0.5)$ multiplications and $O(0.5\log_2 n)$ storage are needed where n is the number of voters. Since the number of voters is often large in voting applications, this is a high cost. To make the search more efficient, the votes may be divided into multiple groups to perform a separate tally in each group. However, this division increases the number of decryptions as a decryption is needed for every candidate in each group.

2.3.2. Mixnets

These systems [9] imitate the process done in conventional elections when, once the voting period is closed, the ballot boxes are shaken in order to shuffle the votes, preventing the correlation of their order with the order in which the voters voted. Once this correlation has been broken, the votes can be decrypted in a secure way in order to obtain the results.

A mixnet is composed of a certain number of mix-nodes where each one receives the encrypted votes from the previous node and permutes them, keeping the permutation in secret so the path of the encrypted votes through the mixnet cannot be recovered.

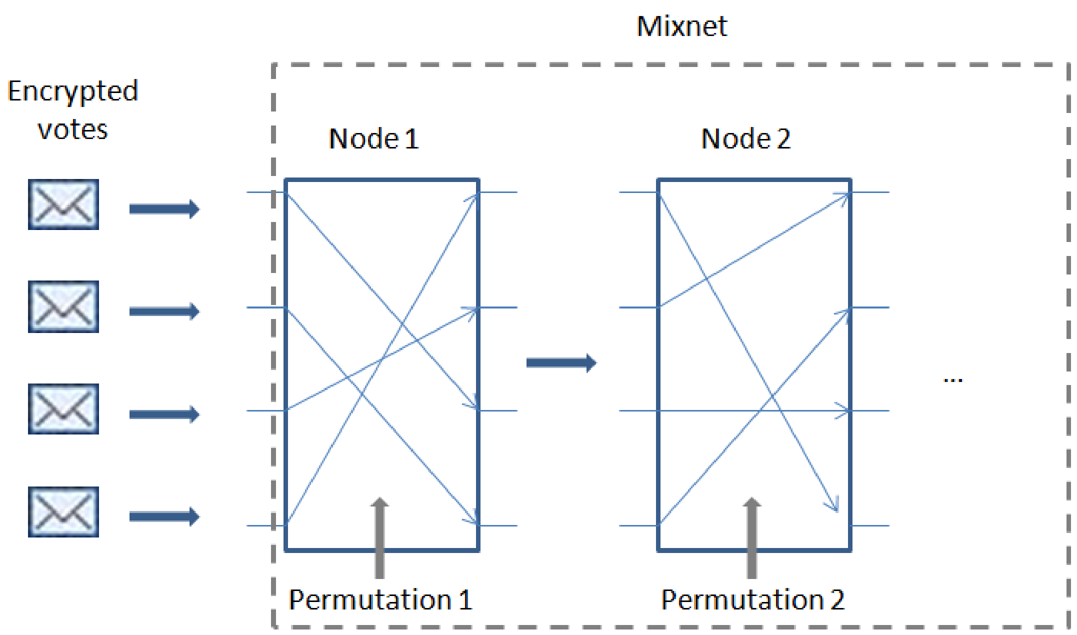


Fig. 2 - 5 Mixnet

This permutation would have no sense if the input and output encrypted votes at each node had the same values, due to that both sets could be compared and the path of each vote could be traced. Therefore, each node does a permutation and a transformation operation over the set of input encrypted votes.

Since the transformation over the encrypted votes modifies their values, appropriate verification methods must be used to ensure the correct behaviour of each node. Otherwise, the nodes could change the contents of their input sets and falsify the election results.

There are a lot of types of mixnets depending on the transformation they do over the encrypted votes and how this transformation is verified:

2.3.2.1. Decryption mixnets

In this type of mixnets the input votes are encrypted in several layers, in such a way that each mix-node has a secret key to remove one of the layers. Therefore, at the output of the last mixnet node the votes are decrypted.

The first proposed mixnet was introduced by Chaum [7]. It was a decryption mixnet that used nested encryption layers, also known as RSA onions with random padding. Mixnets based on this concept of nested encryption and single-layer decryptions at each mix-node are sometimes called “Chaumian mixnets”.
Each mix-node M_i has a public key pk_i and a corresponding secret key sk_i. Inputs to the mixnet are prepared as

$$c = E_{pk_1}\left(r_1, E_{pk_2}\left(r_2, E_{pk_3}\left(r_3, \ldots E_{pk_j}(r_j, m)\right)\right)\right),$$

where *j=1…k* is the number of mix-nodes.

Each mix-node M_i decrypts the outer layer of this onion (if things are done correctly, the outer layer can be decrypted by the designated mix server), removes the random padding r_i, and outputs the resulting set of diminished onions in lexicographic order.

Chaum proposed a variant of this channel for voting protocols, in order to help voters check that their vote was properly forwarded along. The protocol requires two runs of the mixnet: in a first run, the voter sends a public key pk_m for which she has the secret key sk_m and she checks that her public key is present on the final decryption step. Then, in the second run of the mixnet, she sends $(pk_m, E_{sk_m}(m))$ encrypted with the RSA onion, where m is her vote padded with a pre-determined number of 0's. Everyone can perform the public-key based decryption of the vote in the final round, verifying that only 0-padded results emerge.

This two-step mixing ensures that, in the first phase, a voter can complain if her public key is not present in the mixnet output. Then, in the second phase, only messages formed with the first-phase public keys are valid. No one but the sender (in possession of sk_m) can prepare a properly 0-padded plaintext.

This mixnet was broken nine years after its presentation. A message-related attack can be performed in this mixnet, where the attacker can trace the path of a message sending one related to it and searching in the output for two plaintexts with the same relationship. The attack is based on weaknesses of the RSA cryptosystem.

The main drawback of this type of mixnets is that the voter has to encrypt her vote as many times as nodes in the mixnet. That operation can be costly in time and storage resources.

2.3.2.2. Re-encryption mixnets

These mixnets rose from the idea of solving the problem of the onion encryption.

In this type of mixnets encryption algorithms that allow re-encryption are used. Each mix-node re-randomizes the input ciphertexts with fresh randomization values that get algebraically combined with existing randomness, rather than concatenated, in such a way that just a decryption process is needed at the end. Thus, the voter does not need to encrypt the vote as many times as nodes in the mixnet, but once. The re-encryption process results in new values in the output of each node that cannot be related to the ones in the input.

Being a message encrypted in the input of the mixnet

$$c = E_{pk}(m),$$

At each mix-node the message is re-encrypted (R) using the same public key (but using different randomization values

$$c' = R_{pk}(E_{pk}(m))$$

Finally, in the last node, just one decryption process is done to recover the message

$$m = D_{sk}[E_{pk}\left(E_{pk}\left(\ldots E_{pk}(m)\right)\right)]$$

2.3.2.3. Re-encryption with partial decryption mixnets

In this type of mixnets, each node performs a partial decryption of the encrypted vote and a re-encryption.

The messages are encrypted using a public key, such that each mix-node has a share of the public and private key.

In the input of the mixnet, the messages are encrypted as

$$c = E_{pk_1}(m),$$

where pk_1 is the share of the public key for the first mix-node.

Then, the mix-node decrypts the encrypted message with its share of the private key, in such a way that the result is the message encrypted with the share of the public key of the next mix-node

$$D_{sk_1}(m) = E_{pk_2}(m)$$

Finally, the mix-node re-encrypts the message using the public key of the next mix-node in order to avoid a leak of information from the partially decrypted message that could allow an attacker to guess any information to trace the messages in the mixnet

$$E_{pk_2}(D_{sk_1}(m)) = E'_{pk_2}(m)$$

The next mix-nodes perform the same operations, partially decrypting at first and then re-encrypting, until the last node that decrypts and recovers the message.

Several attacks were found in this system based on the semantic security and the malleability of the used cryptosystem. These attacks lead to the definition of ElGamal (the used cryptosystem) for semantic security. This definition is explained in Section 4.

The main issue in the use of these techniques is the verification that the mixing has been performed correctly. Due that each node permutes and transforms the input values (by means of decryption or re-encryption), it is hard to verify that the mix-nodes do not cheat and change the input messages without revealing the permutation or the re-encryption/decryption factor.

Several techniques have been created in order to efficiently verify this mixing process. Some of them are explained in the next section.

2.3.2.4. Universaly verifiable mixnets

Sako and Kilian Mixnet

The first universally verifiable mixnet was introduced by Sako and Kilian [8]. Their mixnet was the first one to provide a proof of correct mixing that any observer can verify.

Their proposal is based on a partial-decryption-and-re-encryption mixnet, where each mix-node publishes the partial decryption results. Then, each node provides a proof of correct partial decryption and a proof of correct re-encryption and shuffling.

The partial decryption proof can be performed using ZK Proofs (Section 5), batched in order to provide a unique proof for all the partially decrypted messages.

The proof of shuffling is a ZK Proof too, where the node first calculates a permutation and re-encryption values to shuffle the messages *(π, $\{r_j\}$)*, and then provides alternative values *(λ,$\{t_j\}$)* to perform a second re-encryption and shuffling according to these new parameters, generating "secondary shuffle outputs". The verifier can then challenge the mix-node to reveal either *(λ,$\{t_j\}$)*, or the relationship between the first and second values: *($\lambda\cdot\pi^{-1}$, $\{r_j-t_j\}$)*, which lets the verifier check how the primary shuffle outputs can be obtained by permuting and re-encrypting the secondary shuffle outputs. With 50% soundness, this is an honest-verifier zero-knowledge proof of correct shuffling.

To increase the assurance of integrity, the mix-node is asked to produce some secondary shuffles. If the node succeeds at responding at *t* challenges, then the primary shuffle is correct with probability *1-2^{-t}*.

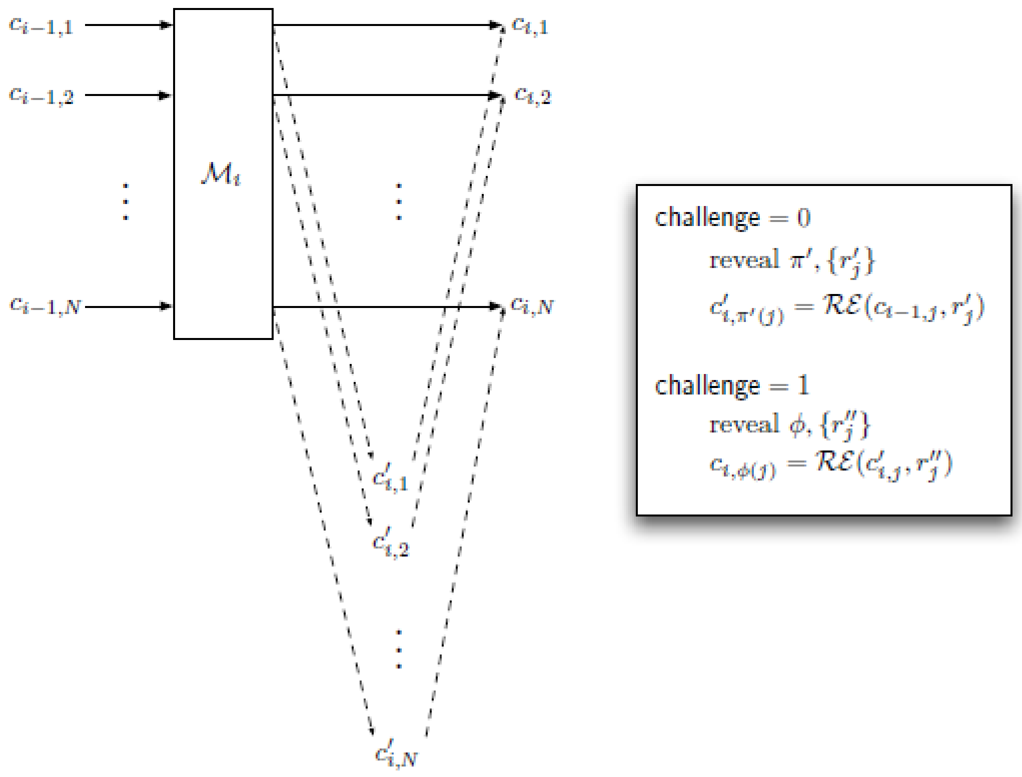


Fig. 2 - 6 Secondary shuffles in the Sako and Kilian proposal for a verifiable mixnet

The main drawback of this system is that a certain number of complex proofs have to be done at each node to verify the correct mixing. The verification cost depends on the number of nodes.

A modification of this mixnet was proposed by Abe [12] where all the mix-nodes generated the secondary shuffles jointly. The mix-node proofs are chained so that the verifier needs only check the output of the last node.

Random Partial Checking

Randomized Partial Checking or RPC in mixnets is a mechanism to check the correct behaviour of the mix nodes. It is based in the basic idea that, rather than providing a proof of completely correct operation, each node provides strong evidence of its correct operation by revealing a pseudo-randomly selected subset of its input/output relations.

The RPC was at first formalized by Markus Jakobsson, Ari Juels and Ronald L.Rivest [10] in 2002 and then enhanced by David Chaum [11]. Nowadays it is still used as a way to prove that a mixnet has mixed correctly its messages without using complex mathematical proofs.

The RPC can be used with Chaumian mixnets where the messages are successively decrypted with each server key, and with mixnets based on a single public key with randomized re-encryption at each layer. It provides voter privacy as a global property of the mixnet rather than as a property ensured by a single honest node.

In an RPC mixnet, the inputs are mixed as usual by a sequence of nodes. The servers then produce strong evidence of their correct operation, revealing partially their input/output relation. For example, a node with *n* inputs can reveal *n/2* inputs randomly without controlling which outputs are selected. This procedure allows for a probabilistic verification of the correct operation of each node.

Privacy is a more delicate affair, since servers will reveal partial information about their input/output relationship.

In a first version [10], paired servers reveal their information in such a way that the paths revealed in the first paired server are not the ones revealed in the second server as it is shown in figure 2.7. Therefore, after the servers reveal partial information, there is still no way to connect any input with a particular output, even if some servers are corrupt.

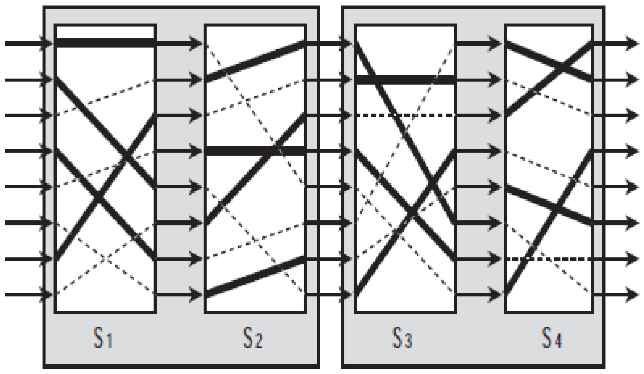


Fig. 2 - 7 Partial disclosure of the node information in RPC

If the first node in a pair reveals half of its input/output pairs, and the other node reveals the complementary half, an adversary that is given complete side information regarding all input/output correspondences for all nodes other than an honest pair can at best identify the corresponding output value with probability 2/n.

In a second version [11], the mixnet nodes are grouped in sequences of four: in the first mixnet node, half of the input/output relationships are chosen to be discovered. In the second mix-node all those not pointed to by those opened in the first mix are discovered. In the third, half of those revealed in the second mixnet and half that not are revealed, and for the fourth node, those not pointed for the ones revealed in the third node are revealed, as it is shown in figure 2.8:

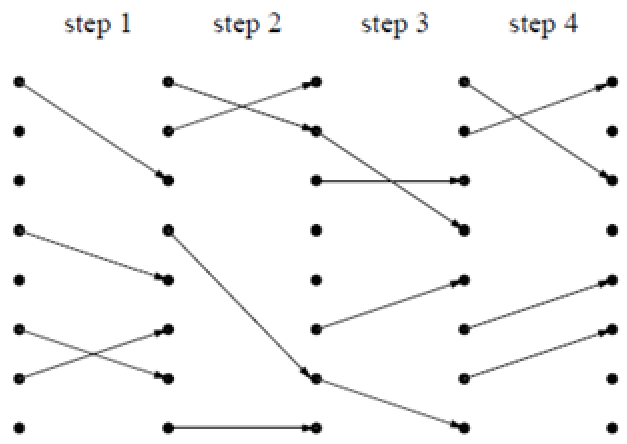


Fig. 2 - 8 Partial disclosure of the node information in Chaum's RPC

In the first version anyone could realise that, after a pair reveals their information, there are outputs for an input message for which the probability of being there is 0 and others for which the probability is 2/n. Otherwise, the probability of an input message of being in a concrete output if information is not revealed is 1/n.

In the second version the problem is fixed because the uniform distribution over *n* is almost achieved: an input message has a 1/n probability of being in a specific output.

During the checking phase each node reveals a fraction *p>0* (usually *p=1/2*) of its input/output correspondences. A node reveals the triple (i, k, R_{ki}), where *k* is the position in the output of the encrypted vote in the position *i* (in the input) for which it has been asked for, and R_{ki} is the re-randomization factor of the re-encrypted vote in this position.

The probability of an adversary to go undetected changing *t* ballots is at most $1/2^t$.

Mixing in small batches

In the late 1990s, the research in mixnets turned to the goal of achieving efficient proofs, where robustness and universal verifiability could be accomplished within practical running times. Rather than using ZK techniques, these new proofs made use of specific number theoretic assumptions of the underlying cryptosystem, usually ElGamal. In particular, where prior proposals required repeating the entire proof to achieve reasonable soundness[1] , these new proposals provided overwhelming soundness in a single round.

Permix [12] and Millimix [13] are an example of mixnets based on using permutation networks that provide better proof speeds for small to medium batches than the prior schemes.

[1] Soundness reflects the probability of a cheater of not being detected. A sound protocol has a high probability of detecting a cheater.

Exponent dot-product Proof

In 2001 Neff [14] introduced what remains to this day the fastest, fully-private, universally verifiable mixnet shuffle proof, requiring 8N exponentiations (where N is the number of votes). This proof is fairly complex.

It is divided in three protocols, at first designed for a mixnet using ElGamal encryption.

- The Equality of Exponent Products proof ensures that both parts of the ElGamal input and output ciphertexts in the mixnet or in a single node have the same random exponent.

- The Known-Exponent Shuffle proof checks using the random exponents that a re-randomization has been done.

- The Proof of Shuffle uses a challenge to make the mix-node demonstrate that there has been a shuffle.

Later, this system was generalized to be used with any homomorphic cryptosystem.

Optimistic Mixing

In 2002, Golle et al. [15] proposed a new type of universally verifiable mixnet they called "Optimistic Mixing", with a significantly faster proof when all players behave honestly. Otherwise, this solution suggests to fallback to existing proofs like Neff's when an error is detected. "Optimistic Mixing" uses ElGamal re-encryption, properly parameterized as per achieve semantic security.

In order to verify integrity before the plaintexts are fully revealed in the last mix-node, two layers of encryption are used.
First the message is encrypted in the usual manner:

$$c = E(m; r) = (a, b) = (g^r, m \cdot h^r)$$

Then, *c* is hashed, and all components are encrypted in a second layer:

$$d = (d_1, d_2, d_3) = \big(E(a; r'), E(b, s'), E(H(a, b); t')\big)$$

The mix-nodes then re-encrypt and shuffle these triples of ciphertexts in the usual manner. In the last node, the first layer is decrypted, obtaining

$$\big(a, b, H(a, b)\big)$$

Then, everyone can verify that the third element is the hash of the other two. If this proof succeeds, the second encryption layer can be removed, recovering *m*.

Another verification process is used combined with this integrity check. Based on the fact that the cryptosystem is homomorphic, the input and output messages at each mix-node are used to calculate input and output proofs that can be compared in order to verify the correct performance of the mixnet.

Concretely, a *proof-of-product* is performed to ensure that the plaintexts of the encrypted messages in the input and in the output of each mix-node are the same.

This *proof-of-product* is the basis of the verification process we will use to evaluate several cryptosystems to be used, so we will explain it broadly in Sections 3 and 4.

Wikström [16] presented several serious attacks against the Optimistic Mixing. These attacks were mainly based on the double envelope encryption process.

In mixing protocols the voter privacy is obtained by means of encrypting the votes and using a shuffling process to decorrelate the encrypted and signed votes and the decrypted ones. Therefore the voter privacy depends on the honesty of the mixing process: if a node does not shuffle the encrypted votes or reveals the permutation and transformation parameters the plaintext ballots could be related to their authors.

Therefore, the main drawback of the mixing techniques is that they are hard to verify in an efficient way.

Another drawback of these systems compared to homomorphic tally protocols is that they are slower, since shuffling and transformation processes are done at each mix-node, and each vote has to be decrypted individually.

Some benefits of these techniques are that they can use more flexible encryption schemes than the ones used in homomorphic tally protocols, they allow the use of write-ins and generally they can be used in more complex elections.

3. Building Blocks

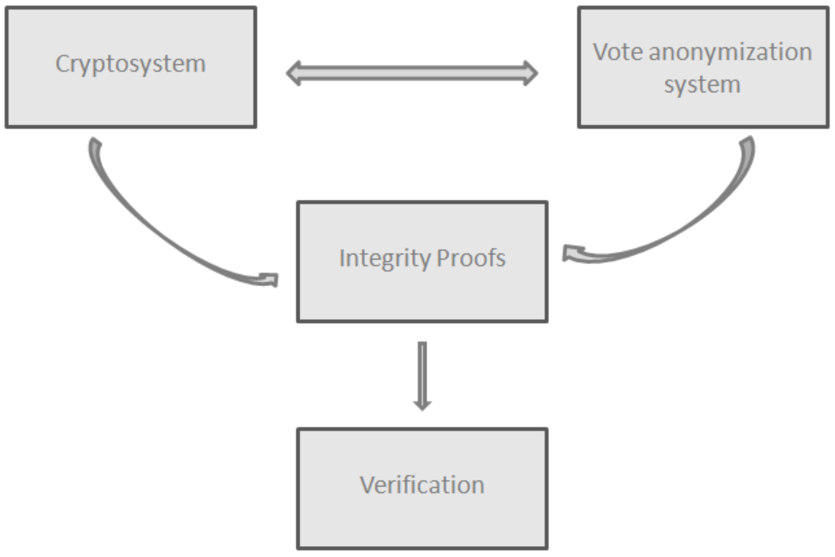


Fig. 3 - 1 Building blocks I

In the previous sections we have explained some advanced cryptographic protocols used in e-Voting environments. Since some of these protocols use cryptosystems with homomorphic properties or/and use ZK proofs to verify their processes, a public key cryptosystem is needed. The main drawback of public key cryptosystems is their inefficiency both in encryption time and in size, so one of the aims of this PFC is to design an efficient asymmetric cryptosystem to be used in e-Voting processes and in other security applications. In order to define the main characteristics of this cryptosystem we should choose a Vote anonymization system to protect voter privacy from the advanced cryptographic protocols explained before.

Since we want this process to be universally verifiable, we want also to be able to calculate some *Integrity Proofs* from the messages in the Vote anonymization system. These *Integrity Proofs* are the basis of the verification process.

In this section we are going to discuss, the building blocks we are going to use and the desired characteristics for them in order to define which protocol must be used or how will it be used.

From the definition of these characteristics we will extract the information to elaborate the requirements for the cryptosystem to be designed.

3.1. Definition of a Vote anonymization system

From the benefits and drawbacks enumerated for the advanced protocols designed to protect voter privacy, we have decided that the best decision is to implement a protocol that implements voter anonymity in the counting phase. Thus, we have to choose if we prefer to use an homomorphic tally protocol or a mixing protocol.

We can resume the benefits and drawbacks of the homomorphic tally and mixing protocols in the following points:

- Homomorphic tally protocols are usually faster in the counting phase than mixnets, since after the encrypted votes are operated just one decryption process has to be performed, instead to the several shuffling and re-encryption/decryption processes done in mixnets.

- Since in an homomorphic tally protocol just one result has to be decrypted, partial decryption techniques using a secret key divided in shares can be applied. Otherwise, when a mixing protocol is used, each vote has to be decrypted individually. The partial decryption using each secret key share for all the votes could be too costly to obtain the results in a reasonable time.

- In homomorphic tally protocols, since the votes are never decrypted individually, it is necessary to ensure that the encrypted votes are well-formed using complex zero knowledge proofs that can slow down the process for the voter and for the voting service. In mixing protocols the correct construction of a vote can be verified when it is decrypted.

- Homomorphic tally protocols can only use homomorphic cryptosystems. Multiple types of cryptosystems can be used in mixnets.

- A vote in an additive homomorphic tally protocol is formed by all the possible options, while in mixnets it just has to contain the selected ones, so the management of the encrypted votes is easier and more efficient.

- The mixing protocols allow the attachment of write-ins in the vote, while the homomorphic tally protocols do not.

- The mixing techniques are hard to verify in an efficient way. Otherwise, the homomorphic tally protocols are easily verifiable (once we have verified all the ZK Proofs).

- The mixing protocols are more efficient for large and complex elections than the homomorphic tally protocols.

We decided to define our vote anonymization system as a mixnet, so we can achieve the management of large elections. Moreover, we want it to be universally verifiable to be able to achieve a high trust level in a real election. Finally, we want this verification to be efficient.

Now, we have to choose the type of mixnet we are going to use:

If we use a decryption mixnet with nested RSA cipher as the one proposed by Chaum, the system is too costly in computation terms in the voter side if the number of nodes in the mixnet is medium or high, since the voter has to encrypt the vote as many times as nodes in the mixnet. Therefore, a decryption mixnet is discarded due to the high cost for the voter.

In our search for a universally verifiable mixnet, we could take the Sako and Killian model for a mixnet with re-encryption and partial decryption. This mixnet uses the specific properties of schemes as Paillier or ElGamal to publish intermediate results in the output of each node, and it is based on the verification of secondary shuffles. These verifications are usually very costly and, as in the case before, the process would be too slow for our purposes.

In his proposal, Neff designs a mixnet with ElGamal encryption. It has been proved that this mixnet is the most secure and verifiable since now, but, although it is the faster one with the highest level of security and verifiability, it is still too slow to perform a mixing in a large-scale election in a reasonable time.

Our aim is to create an efficient cryptosystem based on some ideas of how could be designed a new universally verifiable mixnet that solves the efficiency problems of the existent proposals, where the computation is fast and does not depend on the number of nodes. These ideas are inspired by the Optimistic Mixing proposal explained in Section 2.

In order to achieve this objective we will use what we have called *Integrity Proofs* in a re-encryption mixnet. These proofs are used to verify the correct performance of the mix-nodes in an efficient way.

3.2. Definition of the Integrity Proofs and the Verification process

Before looking at the possible implementations of the encryption scheme to be used in the mixnet we should take a look to the format, composition and performance of the *Integrity Proofs*.

Although these *Integrity Proofs* should be designed to be used in mixnets, they could be used in other environments in order to verify a transformation process done over a set of input messages, giving a set of output messages as result.

The *Integrity Proofs* should have the scope to allow the comparison between the set of input messages in a process (mixnet) and the output ones to verify that there has not been any manipulation, comparing the contents of the input and output sets without revealing the content of each message on its own.

To achieve these requirements, the homomorphic encryption schemes are very useful:

In an homomorphic cryptosystem the operation upon one or some encrypted messages results in an encryption of the operation of the messages: $E(m_A)\Theta\, E(m_B)= E(m_A \Phi\, m_B)$.

Depending on the encryption algorithm, the homomorphism can be additive or multiplicative. That is, for example, $E(m_A)\cdot E(m_B)= E(m_A+ m_B)$, or $E(m_A) \cdot E(m_B)= E(m_A \cdot m_B)$ respectively.

Therefore, using an homomorphic encryption scheme we can generate *Integrity Proofs* that allow the comparison of the mixnet input set of encrypted messages and the set of output plaintexts:

- An *Input Integrity Proof* is calculated operating the set of input encrypted messages.

- Once the messages have been decrypted an *Output Integrity Proof* is generated operating the set of decrypted messages.
- The *Input Integrity Proof* is decrypted.
- The decrypted *Input Integrity Proof* and the *Output Integrity Proof* are compared.
- If both values match, that means that the mixing has not altered the overall integrity of the votes.

The figure 3.2 shows the verification process based on these *Integrity Proofs*.

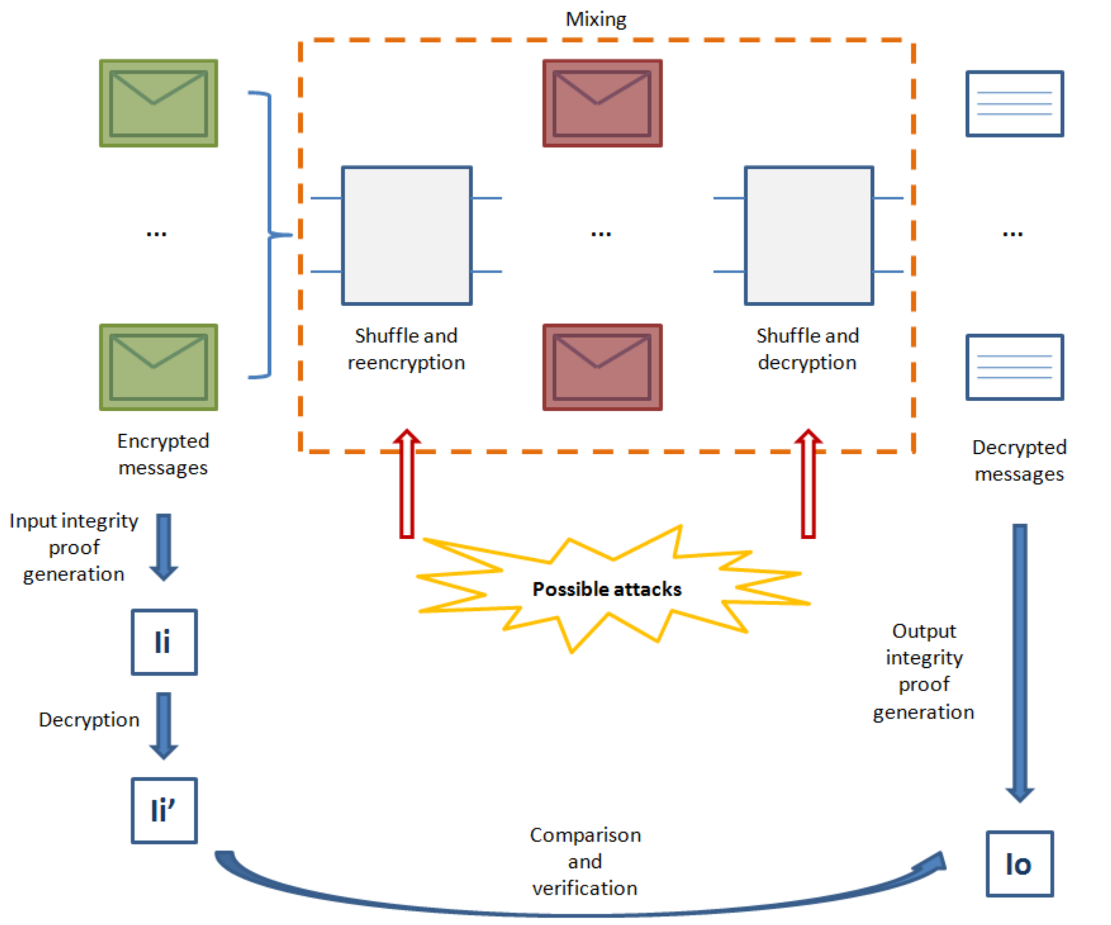


Fig. 3 - 2 Verification based on *Integrity Proofs*

3.3. Cryptosystem Definition

In accordance with the basic requirements explained above, the cryptosystem to use in this environment should have homomorphic properties and be suitable to be used in a re-encryption mixnet.

The building blocks can now be defined more accurately as:

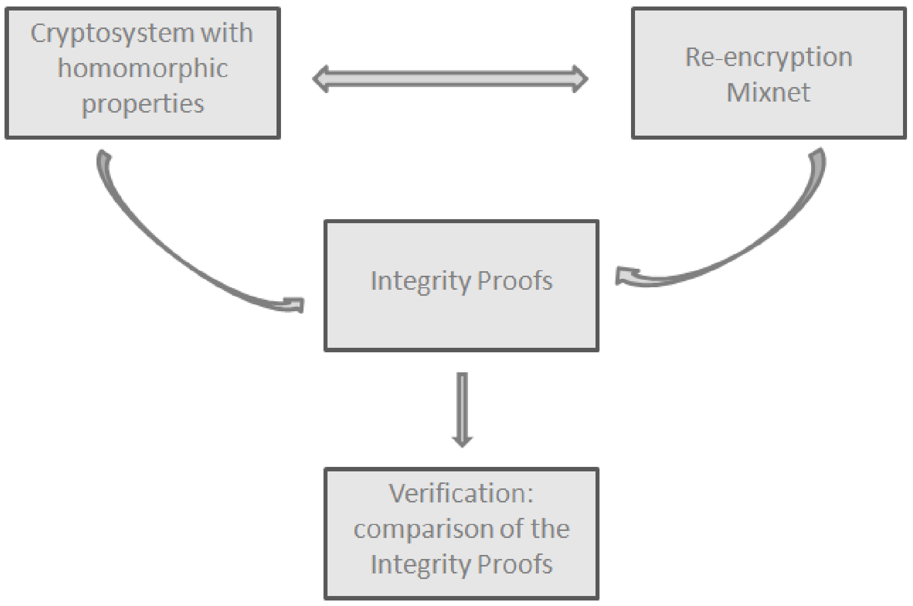

Cryptosystem with homomorphic properties
Re-encryption Mixnet
Integrity Proofs
Verification: comparison of the Integrity Proofs

Fig. 3 - 3 Building blocks II

Finally, some cryptosystems with these properties are presented here. In the next section, one of these cryptosystems will be chosen to be the basis of a new encryption scheme.

3.3.1. RSA

The RSA cryptosystem, named after its inventors R. Rivest, A. Shamir, and L. Adlema, is the most widely used public-key cryptosystem. It may be used to provide both secrecy and digital signatures and its security is based on the intractability of the integer factorization problem for large numbers.

3.3.1.1. Key generation

Each entity creates an RSA public key and a corresponding private key. Each entity *A* should do the following:

- Generate two large random and distinct primes p and q, each roughly the same size.
- Compute $n=pq$ and $\varphi=(p-1)(q-1)$.
- Select a random integer e, $1<e<\varphi$, such that $gcd(e, \varphi)=1$.
- Use the extended Euclidean algorithm [17] to compute the unique integer d, $1<d<\varphi$, such that $ed\equiv 1 \bmod \varphi$.
- *A*'s public key is (n, e); *A*'s private key is d.

The recommended key size (size of n and d) is 2048 bits [18].

3.3.1.2. RSA public key encryption

B encrypts a message *m* for *A*, which *A* decrypts:

- Encryption: *B* should do the following:
  - Obtain *A*'s public key *(n, e).*
  - Represent the message as an integer *m* in the interval *[0, n-1]*.
  - Compute $c = m^e \ mod \ n.$
  - Send the ciphertext *c* to *A*.
- Decryption: to recover the plaintext *m* from *c*, *A* should do the following:
  - Use the private key *d* to recover $m = c^d \ mod \ n.$

Proof that the decryption works:

Since *ed≡1 mod φ*, there is an integer *k* such that *ed=1+kφ*. Now, if *gcd(m, p)=1* then by Fermat's theorem,

$$m^{p-1} \equiv 1 \ mod \ p.$$

Raising both sides of this congruence to the power *k(q-1)* and then multiplying both sides by *m* yields

$$m^{1+k\cdot(p-1)\cdot(q-1)} \equiv m \ mod \ p.$$

On the other hand, if *gcd(m ,p)=p*, then this last congruence is again valid since each side is congruent to 0 modulo *p*. Hence, in all cases

$$m^{ed} \equiv m \ mod \ p.$$

By the same argument,

$$m^{ed} \equiv m \ mod \ q.$$

Finally, since *p* and *q* are distinct primes, it follows that

$$m^{ed} \equiv m \ mod \ n,$$

and, hence,

$$c^d \equiv (m^e)^d \equiv m \ mod \ n.$$

Encryption and decryption exponents

In order to improve the efficiency of encryption, it is desirable to select a small encryption exponent *e*. It may seem desirable to select a small decryption exponent *d* too in order to improve the efficiency of decryption. However, if *d* is small, there are efficient algorithms for computing *d* from the public information *(n, e)*. To avoid this attack, the decryption exponent *d* should be roughly the same size as *n*.

Homomorphic properties

Let m_1 and m_2 be two plaintext messages, and let c_1 and c_2 be their respective RSA encryptions. Observe that

$$(m_1 \cdot m_2)^e \equiv {m_1}^e \cdot {m_2}^e \equiv c_1 \cdot c_2 \ mod\ n,$$

so RSA has multiplicative homomorphic properties.

Adaptive chosen-ciphertext attack

The homomorphic properties of RSA lead to the Adaptive chosen-ciphertext attack: suppose that an active adversary wishes to decrypt a particular ciphertext $c = m^e\ mod\ n$ intended for *A*. Suppose also that *A* will decrypt arbitrary ciphertexts for the adversary, other than *c* itself. The adversary can conceal *c* by selecting a random integer $x \in Z_n^*$ and computing $c' = c \cdot x^e\ mod\ n$. Upon presentation of *c'*, *A* will compute for the adversary $m' = c'^d\ mod\ n$. Since

$$m' \equiv c'^d \equiv c^d \cdot (x^e)^d \equiv m \cdot x\ mod\ n,$$

the adversary can then compute $m = m' \cdot x^{-1}\ mod\ n$.

This adaptive chosen-ciphertext attack should be circumvented in practice by imposing some structural constraints on plaintext messages.

A usual practice is to add a random padding to the message to be encrypted. This increases the size of the message, which guarantees that the encrypted message is large enough that it will not be easy to use for an attack, and it adds pseudo-random information in a way that means that a given plaintext message could be encrypted to a wide range of different ciphertexts (probabilistic encryption), depending on the choices made during padding (since RSA is a deterministic cryptosystem the same message encrypted twice results in the same two ciphertexts).

However, when RSA is used with a padding scheme the cryptosystem is not homomorphic any more.

$$((m_1|P_1) \cdot (m_2|P_2))^e \neq (m_1|P_1)^e \cdot (m_2|P_2))^e \neq c_1 \cdot c_2 \ mod\ n$$

3.3.2. ElGamal

The public key ElGamal cryptosystem can be viewed as a Diffie-Hellman key agreement protocol in a key transfer mode. Its security is based on the intractability of the discrete logarithm problem and the Diffie-Hellman problem.

3.3.2.1. *ElGamal key generation*

- Generate a prime large random number *p* and a generator *g* of the module *p* integer multiplicative group Z_p*.

- Choose a random integer *x* where $2 \leq x \leq p-2$ as the secret key.

- Then calculate $h=g^x \bmod p$.

- Finally, the public key is *(p, g, h)*, and the private key is *x*.

3.3.2.2. *ElGamal encryption*

B must do the following to encrypt a message for *A*:

- Obtain *A*'s public key *(p, g, h)*.

- Represent the message as an integer *m* in the range *(0, 1, ..., p-1)*.

  Select a random integer *r*, where $1 \leq r \leq p-2$.

- Compute $\gamma = g^r \bmod p$, and $\Phi = m \cdot h^r \bmod p$.

- Send the ciphertext *c=(γ, Φ)* to *A*.

A must follow these steps to decrypt the message:

- Use *A*'s private key to compute $\gamma^{-x} = g^{-xr}$.

- Recover *m* as $m = \Phi \cdot \gamma^{-x}$.

The proof that the decryption works is:

$$\Phi \cdot \gamma^{-x} = m \cdot (g^x)^r \cdot g^{-xr} = m$$

Since in a shared system all the parties use the same prime *p* and the generator *g*, *p* and *g* do not need to be published with the public key because they are already known. This contributes to the key size reduction. An additional advantage of having a fixed base *g* is that the exponentiations can be calculated in previous steps of the protocol. A disadvantage of these shared systems is that *p* has to be very large to preserve the cryptosystem security (nowadays 1024 or 2048 bits is recommended).

Multiplicative homomorphism:

The ElGamal encryption, as RSA has multiplicative homomorphic properties:

Being two messages m_A and m_B encrypted as

$$C_A = (m_A \cdot h^{r_A}, g^{r_A})$$

$$C_B = (m_B \cdot h^{r_B}, g^{r_B}),$$

the multiplication of both ciphertexts

$$C_A \cdot C_B = (m_A \cdot m_B \cdot h^{r_A + r_B}, g^{r_A + r_B})$$

is equal to the encryption of the result of $m_A \cdot m_B$, so

$$E(m_A) \cdot E(m_B) = E(m_A \cdot m_B).$$

Additive homomorphism:

A modification of ElGamal cryptosystem, called exponential ElGamal, is used when additive homomorphic properties are used. In this modification a new parameter λ is used to exponentiate the message.

In exponential ElGamal a message is encrypted as

$$c = (\lambda^{\mathrm{m}} \cdot h^r, g^r).$$

Being two messages m_A and m_B encrypted as

$$C_A = (\lambda^{\mathrm{m_A}} \cdot h^{r_A}, g^{r_A})$$

$$C_B = (\lambda^{\mathrm{m_B}} \cdot h^{r_B}, g^{r_B}),$$

the multiplication of both ciphertexts

$$C_A \cdot C_B = (\lambda^{\mathrm{m_A + m_B}} \cdot h^{r_A + r_B}, g^{r_A + r_B})$$

is equal to the encryption of the result of $m_A + m_B$, so

$$E(m_A) \cdot E(m_B) = E(m_A + m_B).$$

Since ElGamal is a probabilistic encryption scheme, padding schemes are not used in this cryptosystem.

3.3.3. Paillier

The Paillier cryptosystem, named after and invented by Pascal Paillier in 1999, is a probabilistic asymmetric algorithm for public key cryptography. The decisional composite residuosity assumption is the intractability hypothesis upon which this cryptosystem is based.

The decisional composite residuosity assumption states that, given a composite *n* and an integer *z*, it is hard to decide whether *z* is a n-residue modulo n^2 or not, i.e., whether there exists *y* such that $z=y^n(mod\ n^2)$.

3.3.3.1. Key generation

- Choose two large prime numbers *p* and *q* randomly and independently of each other, such that $\gcd\big(p \cdot q, (p-1)\cdot(q-1)\big) = 1$. This property is assured if both primes are of equivalent length.

- Compute $n = p \cdot q$ and $\lambda = lcm(p-1, q-1)$.

- Select a random integer *g*, where $g \ \in Z^*_{n^2}$.

- Ensure that *n* divides the order of *g* by checking the existence of the following modular multiplicative inverse:

$$\mu = \left(L\left(g^{\lambda}\ mod\ n^2\right)\right)^{-1}\ mod\ n,$$

  where $L(u) = \frac{u-1}{n}$

- The public key is *(n, g)*.

- The private key is *(λ, u)*.

If using *p* and *q* of equivalent length, a simplified variant of the above key generation steps would be to set $g = n+1, \lambda = \ \varphi(n)$, and $\ \mu = \ \varphi(n)^{-1}\ mod\ n$, where $\varphi(n) = (p-1)\cdot(q-1)$.

3.3.3.2. Encryption

B must do the following to encrypt a message for *A*:

- Let *m* be a message to be encrypted where $m \ \in Z_n$.

- Select random *r* where $r \in Z^*_n$.

- Compute the ciphertext as

$$c = g^m \cdot r^r\ mod\ n^2$$

- Send the ciphertext *c* to *A*.

A must follow these steps to decrypt the message:

- Recover *m* as $m = L\left(c^{\lambda}\ mod\ n^2\right) \cdot \mu\ mod\ n.$

Homomorphic properties

A notable feature of the Paillier cryptosystem is its homomorphic properties. Sinces the encryption function is additively homomorphic, the following identities can be described:

<u>Homomorphic addition of plaintexts:</u>

The decryption of the product of two ciphertexts returns the sum of their corresponding plaintexts:

$$D(E(m_1,r_1) \cdot E(m_2,r_2) \; mod \; n^2) = \; m_1 + m_2 \; mod \; n.$$

<u>Homomorphic multiplication of plaintexts:</u>

The decryption of an encrypted plaintext exponentiated to another plaintext returns the product of the two plaintexts:

$$D(E(m_1,r_1)^{m_2} \; mod \; n^2) = \; m_1 \cdot m_2 \; mod \; n$$

$$D(E(m_2,r_2)^{m_1} \; mod \; n^2) = \; m_1 \cdot m_2 \; mod \; n$$

More generally,

$$D(E(m_1,r_1)^{k} \;\; mod \; n^2) = \; m_1 \cdot k \; mod \; n$$

However, given the Paillier encryptions of two messages there is no known way to compute an encryption of the product of these messages without knowing the private key.

The original cryptosystem does provide semantic security against chosen-plaintext attacks. The key length in Paillier is the same as in ElGamal, being recommended for *n* to have at least 1024 bits, better 2048.

4. Design of a Cryptosystem for an Efficient and Universally Verifiable Mixnet

The main objective of this PFC is to design an efficient cryptosystem that could be used in e-Voting environments in order to encrypt the votes to be processed by a re-encryption mixnet that uses universal verification techniques to ensure the correct performance of the mixing process.

Therefore, the cryptosystem is designed in order to be used in this context, although it can be used generally in other environments where a message needs to be encrypted at first and then verifiably decrypted.

The model of a re-encryption mixnet where *Integrity Proofs* are used to achieve universal verifiability is used as a case study where the different proposals are evaluated.

4.1. Initial Assumptions

To design our cryptosystem we first enumerate a series of initial assumptions in order to clarify which are our main objectives, requirements, preferences and priorities:

- We want to generate *Integrity Proofs* in order to verify the correct behaviour of a re-encryption mixnet. These *Integrity Proofs* are generated from operations based on the message content. To compare the content of the messages in the input of the mixnet with the ones in the output using Integrity Proofs, a homomorphic encryption scheme is needed. The homomorphic property of a cryptosystem is explained in the sections *Integrity Proofs* (3.2) and *ElGamal Homomorphisms* (4.3.2).

  ElGamal, RSA, Paillier, or Elliptic Curve cryptosystems based on the Discrete Logarithm Problem usually have homomorphic properties.

- Since the re-encryption mixnets are less costly for the voter than the decryption ones, we need to use cryptosystems with suitable properties for the performance of the first ones. In these cryptosystems the messages are encrypted in such a way that, when various re-encryption processess are done, it is just necessary one decryption step to get the plaintext messages. At each re-encryption step the ciphertexts are re-randomized changing their values, so probabilistic encryption algorithms must be used.

  Cryptosystems like ElGamal, Paillier, or Elliptic Curve cryptosystems based on the Discrete Logarithm Problem are suitable to be used in this type of mixnets.
  RSA is a deterministic encryption algorithm, meaning that when a ciphertext is re-encrypted using the same public key, the value is the same than before the re-encryption (an undesired property in re-encryption mixnets). This problem can be solved using padding at each re-encryption step, but then the cryptosystem loses its homomorphic properties.

- Zero-Knowledge Proofs are usual in advanced cryptographic applications, i.e. to proof that a part of a protocol proceeds properly without revealing important or confidential information. In order to achieve the maximum verifiability level in the mixing process some of these ZK Proofs have to be used, so the cryptosystem must have the option of calculating them.

  In our research we have found that the Paillier encryption scheme does not have the ability to calculate this type of proofs (although there have been some advances in this direction, while ElGamal has a broad range of ZK Proofs that we can use.

- In an e-Voting environment all the votes cast by the voters are encrypted using the same public key, so just a trustable entity can decrypt them. In order to enhance the system security, the secret key needed to decrypt the votes can be split and shared among several different parties, in such a way that just a majority of them can recover the secret key. The system security level is increased if, instead of splitting the secret key, it is generated in a distributed way among all the parties, so the shares are generated but the entire key is not.

  The ElGamal cryptosystem is more suitable to be used in Distributed Key Generation algorithms than Paillier.

Taking account of the initial assumptions discussed above, the cryptosystems that are considered to fulfil them are ElGamal and EC-ElGamal in case of working with Elliptic Curve Cryptography.

4.2. Analysis Model

In order to design a new cryptosystem to be used in a verifiable mixnet, we need an analysis model where we can consider the possible protection steps that we think that are necessary in order to guarantee the system security.

This scheme specifies the message processing steps and the possible entry points for an attacker. Then, when analyzing our proposals, we are going to considerate these steps in order to evaluate the scope of an attacker at each entry point and how to prevent or mitigate its effects.

The analysis model is presented in the figure 4.1.

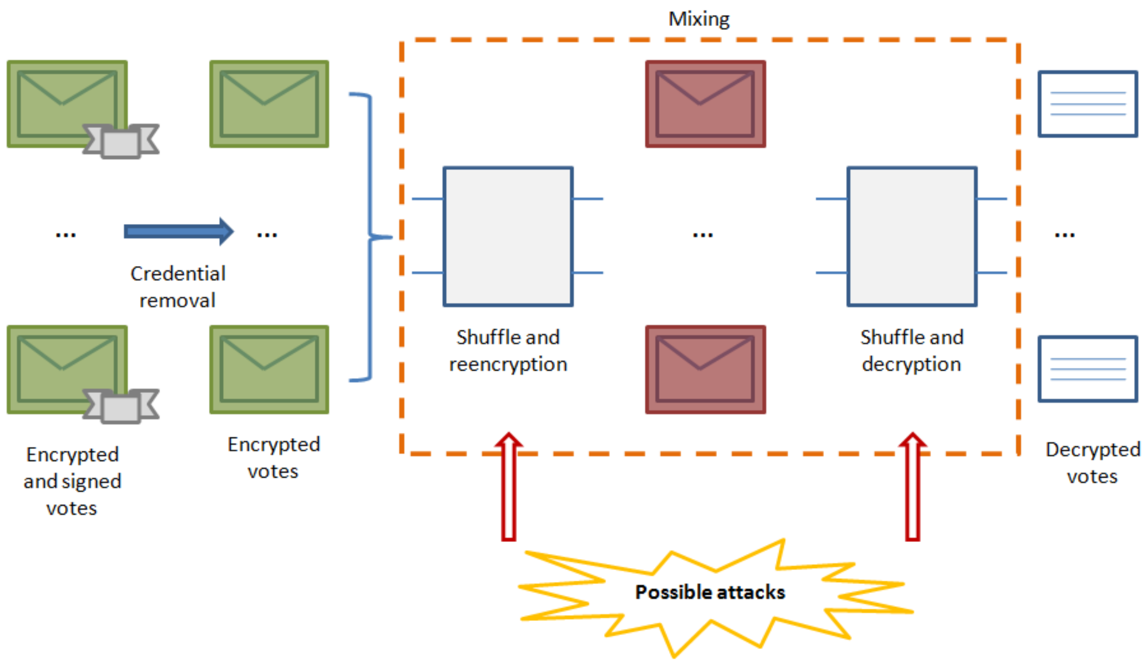


Fig. 4 - 1 Analysis model

In the analysis model the voters cast their encrypted votes to the voting service that stores them. After the voting process is closed, the received votes arrive to the mixing service encrypted and signed. The purpose of the signature is to identify the elegibility of the voter that casts the vote, and to check the vote integrity. Once the verifications are done, these signatures are removed, so that the encrypted votes that enter in the mixnet are not connected to their issuers anymore.

The mixnet has two nodes. It is a re-encryption mixnet where the first node shuffles and re-encrypts the votes and the second one shuffles and decrypts them. An attacker could then have an entry point to manipulate the votes in each node, since their operations are kept secret in order to preserve the voter privacy and therefore their procedures are hard to verify.

4.3. ElGamal Cryptosystem

Based on the fact that:

1. We need a homomorphic encryption scheme in order to calculate *Integrity Proofs*.
2. We want to use Zero Knowledge Proofs to verify the correct performance of the system, i.e. the correct decryption.
3. We have to use a suitable cryptosystem for re-encryption mixnets.

Our choice of a cryptosystem as a basis of the design of a new one suitable for our purposes is the ElGamal cryptosystem.

4.3.1. Conventional ElGamal encryption

4.3.1.1. ElGamal key generation

- Generate a prime large random number p and a generator g of the module p integer multiplicative group Z_p*.
- Choose a random integer x where $2 \leq x \leq p-2$ as the secret key.
- Then calculate $h=g^x$ mod p.
- Finally, the public key is (p, g, h), and the private key is x.

4.3.1.2. ElGamal encryption

B must do the following to encrypt a message for *A*:

- Obtain *A*'s public key (p, g, h).
- Represent the message as an integer m in the range $(0, 1, ..., p-1)$.

  Select a random integer r, where $1\leq r\leq p-2$.
- Compute $\gamma=g^r \bmod p$, and $\Phi= m\cdot h^k \bmod p$.
- Send the ciphertext $c=(\gamma, \Phi)$ to *A*.

A must follow these steps to decrypt the message:

- Use *A*'s private key to compute $\gamma^{-x}=g^{-xr}$.
- Recover m as $m= \Phi \cdot \gamma^{-x}$.

4.3.1.3. Key generation

In the first step, a prime p is selected. It is recommended for p to be at least of 1024 bits length (this is the size of the currently used ElGamal keys), or preferably 2048. It is usual to choose p as a safe prime [19] [20] in order to protect the cipher against the Polling-Hellman algorithm to calculate discrete logarithms [Definition 4.85 in 17]. To calculate p as a safe prime, $p=2q+1$, where q is also prime [Algorithm 4.86 in 17].

4.3.1.4. Message encryption

In order to encrypt a message, it is necessary to convert it in a number, for example using ANSI code. If the message is too large, it has to be divided in blocks, in a way that each block length is at most $p-1$.

4.3.1.5. *Cryptosystem characteristics*

- Efficiency: the encryption process needs two modular exponentiations: $g^x \bmod p$ and $g^{xr} \bmod p$. Is important to have an exponent large enough to avoid a search using the baby-step giant-step algorithm [21], so these exponentiation operations may be computationally costly.

  A disadvantage is that the encryption has an expansion factor of two (at least). An encrypted message length is the double of the plaintext one.

  A message with the same length as *p-1* (the maximum length) doubles its size when it is encrypted, since the ciphertext has two components, g^r and $m \cdot h^k$, as large as *p-1*. In case that the message is smaller the expansion factor is higher.

- Probabilistic encryption: the ElGamal cryptosystem uses randomization at the encryption process. The main idea behind these practices is to use the randomness to increase the cryptographic security of a cipher process through one or some of these methods:

  - Increasing the effective space size of the clear texts.

  - Decreasing the effectiveness of chosen-plaintext attacks by virtue of a one-to-many mapping of plaintext to ciphertext.

  - Decreasing the effectiveness of statistical attacks by levelling the a priori probability distribution of inputs.

- Security: the problem breaking the ElGamal cryptosystem is equivalent to solve the Diffie-Hellman problem. In fact, the ElGamal algorithm can be viewed as a message encryption multiplying it with the Diffie-Hellman [21] session key g^{xr}.

  It is important to use different random exponents when a new message has to be encrypted, in order to prevent the break of an encryption based on the knowledge of one ciphertext and plaintext couple.

4.3.2. ElGamal homomorphisms

Multiplicative homomorphism

The ElGamal cryptosystem is homomorphic. Therefore, as it has already been explained, the operation upon some encrypted messages results in the encryption of the operation of the messages: $E(m_A)\Theta\, E(m_B)= E(m_A \Phi\, m_B)$.

Depending on the encryption algorithm, the multiplication of the encrypted messages can be equal to an additive homomorphism $E(m_A)\cdot E(m_B)= E(m_A+ m_B)$ or a multiplicative one $E(m_A) \cdot E(m_B)= E(m_A \cdot m_B)$.

The ElGamal cryptosystem has natively multiplicative homomorphic properties:

Being a message m_A encrypted as $C_A = (m_A \cdot h^{r_A}, g^{r_A})$ and a message m_B encrypted as $C_B = (m_B \cdot h^{r_B}, g^{r_B})$, the multiplication of both ciphertexts $C_A \cdot C_B = (m_A \cdot m_B \cdot h^{r_A + r_B}, g^{r_A + r_B})$ is equal to the encryption of the result of $m_A \cdot m_B$, so

$$E(m_A) \cdot E(m_B) = E(m_A \cdot m_B)$$

Additive homomorphism

A variation of the ElGamal cryptosystem, usually named exponential ElGamal cryptosystem, provides additive homomorphic properties. In the exponential ElGamal cryptosystem another public parameter $\lambda \in \{1, p-1\}$ is used to exponentiate the message.

Being a message m_A encrypted as $C_A = (\lambda^{m_A} \cdot h^{r_A}, g^{r_A})$ and a message m_B encrypted as $C_B = (\lambda^{m_B} \cdot h^{r_B}, g^{r_B})$, the multiplication of both ciphertexts $C_A \cdot C_B = (\lambda^{m_A + m_B} \cdot h^{r_A + r_B}, g^{r_A + r_B})$ is equal to the encryption of the result of $m_A + m_B$, so

$$E(m_A) \cdot E(m_B) = E(m_A + m_B)$$

This latter option is usually used in voting systems, where the message *m* is represented as a one-bit choice that means a vote for a single option [22][14][5], where *m=1* means an affirmative answer and *m=0* is a negative one. Then, a homomorphic tally is performed, multiplying all the encrypted votes to get the result of their addition. The benefits and drawbacks of these techniques are explained in Section 2.

4.3.3. Definition of ElGamal for semantic security in voting applications

The semantic security for an encryption scheme can be defined based on indistinguishability. If it is infeasible for an adversarial algorithm to distinguish between the encryptions of any two messages, even if these messages are given, then the encryption should not reveal any information about the messages. In other words, given m_A, m_B and $c_A = E(m_A), c_B = E(m_B)$, where *E()* denotes an encryption operation, an attacker cannot succeed choosing which is the ciphertext corresponding to m_A with a probability higher than 0,5.

Attack based on semantic security

ElGamal used over all of *Zp** is not semantically secure by default: by testing the respective subgroup memberships of $g^r$and $m \cdot h^r$, one can infer information about the subgroup membership of *m*. Based on this idea, *Birgit Pfitzmann* [23] found a passive attack on the first re-encryption mixnet [24].

The idea of the attack is basically that the encryption scheme used does not hide all partial information, and thus one can restrict the number of participants who may have sent a particular message. More precisely, the group *Zp** has subgroups. At least the group order *p-1* is even, and thus there is a subgroup G_2 of order 2, generated by g^2.

Anybody can easily test if a group element *a* is in G_2 by testing if:

$$a^{(p-1)/2} = 1 \bmod p$$

Similarly, if *p-1* has another prime factor f, there is a subgroup G_f, and the criterion for membership in G_f is:

$$a^{(p-1)/f} = 1 \; mod \; p$$

The residue class of a message with respect to such a subgroup is not hidden. For example, in a mixnet with just one mix-node and two participants P_1 and P_2 with messages m_1 and m_2, where all three are honest, an outsider should not be able to trace which of the two participants sent which of the two messages (as we have defined semantic security based on indistinguishability). The two ciphertexts are:

$$C_1 = (t_1, u_1) = (g^{r_1}, m_1 \cdot h^{r_1})$$
$$C_2 = (t_2, u_2) = (g^{r_2}, m_2 \cdot h^{r_2})$$

Note that it is publicly known that C_1 was sent by P_1 and C_2 by P_2. The mix node outputs m_1 and m_2 shuffled. However, the attacker can test if t_1 is in the subgroup G_2. This is true in half the cases. If so, he knows that h^{r_1} is in this subgroup too, since

$$h^{r_1} = (g^x)^{r_1}$$

So he knows that $u_1 \epsilon \; G_2 \leftrightarrow m_1 \epsilon \; G_2$ and if only one of m_1 and m_2 is in the subgroup, the attacker knows which of them is P_1's message.

If there are more than two participants, the attacker can partition them into possible senders of certain messages. If there is more than one mix-node, the attacker can relate the initial C_j and the final m_j in the same way.

We can conclude that in a re-encryption-based mixnet, if the cryptosystem is not semantically secure, an attacker can detect input/output relationships.

Countermeasure

The countermeasure against the simple passive attack is to use a multiplicative group of prime order. Usually one does this by choosing two prime numbers: *p* as a safe prime and *q* so that *q|p-1*, and selecting g as a generator of a *q*-order subgroup of *Zp**, G_Q. These numbers and group generator are selected in a way that makes given messages encoded into this subgroup easy to recover. On usual way is to use *p=2q+1*.

Therefore, the messages to be encrypted are of order Q, and the random exponents to encrypt are in *Zq*.

*Zp** can be split in two main subgroups based on having integers that are quadratic residues modulo *p* or not. An integer *a* is a quadratic residue modulo *p* if there is a number *c* so that $c^2 \equiv a \; mod \; p$. In order to determine if *a* is a quadratic residue modulo *p* the Legendre symbol $\left(\frac{a}{p}\right)$ is evaluated:

- $\left(\frac{a}{p}\right)$ = 1 if *a* is a quadratic residue modulo *p*,
- $\left(\frac{a}{p}\right)$ = -1 if *a* is a quadratic non-residue modulo *p*,
- $\left(\frac{a}{p}\right)$ = 0 if *p|a*.

The relevance of having messages that are quadratic residues in the encryption process is related to the behaviour of these integers when multiplied by the public key *h* or an exponentiation of it (i.e. the encryption factor h^r). If h^r is a quadratic residue and the message is also a quadratic residue, the result is always a quadratic residue. Otherwise, the result is always a non-quadratic residue.

Since the quadratic residuosity of *h* and a ciphertext is easy to test, it is also easy to discern if the original encrypted message related to the ciphertext is quadratic residue modulo *p* or not. To prevent this issue from happening, it is recommended to choose all values representing any of the voting options (or all the messages to be used in the same system) from the quadratic residue group or the quadratic non-residue one to make the ciphertexts indistinguishable.

In order to simplify the notation, from now, all the operations are defined in *Zp*, the messages are chosen from the group *Gq* (all quadratic residues or quadratic non-residues modulo *p*), and the random numbers in the exponents are chosen in *Zq*.

4.4. Proposals for a new cryptosystem based on ElGamal encryption

At first, a list of requirements has been done in order to evaluate which is the most suitable cryptosystem to use in an e-Voting environment using re-encryption mixnets. Once we have found that the ElGamal cryptosystem is the one that fits all our first requirements, we should think in some other characteristics that would be useful for our purposes, like size or speed characteristics.

For example, in a conventional encryption scheme the length of a message has to be at most *p-1*. If the message is larger, it is divided in blocs that are encrypted individually. This is a non-efficient solution, due to the fact that each encrypted message at least doubles its original size, since the ciphertext has two components, g^r and $m{\cdot}h^k$, as large as *p-1*.

Therefore, one of the improvements that we want to achieve in our proposal for a new cryptosystem based on ElGamal encryption is that the long messages are encrypted more efficiently.

Some characteristics desired for the encryption scheme are:

- Since this cryptosystem is designed to be used in a re-encryption mixnet, it must be infeasible to reconstruct the encrypted message from the one decrypted after re-encryption, so the content can not be related to the sender.

- Neither of the parts of the ciphertext can remain constant through the re-encryption process, since then the output and input messages in the mixnet could be connected.

- The verification of the decryption process does not compromise the voter privacy.

- The cryptosystem must have homomorphic properties to calculate *Integrity Proofs* in order to detect message manipulation.

- It must support large messages without prefixed content (as write-ins).

- If a new multi-block message encryption scheme (for large messages) is designed, it has to accomplish that:

- It must allow re-encryption.
- The homomorphic characteristics of ElGamal encryption has to be preserved in order to calculate *Integrity Proofs*.
- If the content of one block of a message is disclosed, it must be infeasible to discover the content of other blocks of the same message.

Analysis of existent proposals and design of new ones

In order to study and design new proposals, we will focus first on achieving a more efficient encryption for long messages. In e-Voting environments one could consider that there is no need of large contents management, because the choices to be encrypted are usually short texts (i.e. candidate names), but we could consider the different choices of a voter as blocks of the same message that has to be encrypted (i.e. when additive homomorphic cryptosystems are used to perform homomorphic tally). Then, an efficient way to encrypt these different choices would be useful.

We have talked about the size problem when encrypting large messages, due that each block at least doubles its size when encrypted. But we have to talk about performance too. Talking about time cost, for each message block m_i a new random r_i must be generated and two exponentiations for h^{ri} and g^{ri} have to be done. The exponentiation is an operation that has a great computational cost in encryption contexts; therefore other methods have been studied to encrypt block-divided messages in a more optimistic way, both in performance and size.

First, we are going to analyze some academic proposals that have been designed to enhance the ElGamal encryption performance for long messages. Then we are going to explain our proposals for an encryption system in order to meet our requirements.

In order to compare the computational effectiveness of each proposal, we will consider in a first place the effort (in time) needed to encrypt or decrypt a message. Since the most costly computations in these cryptosystems are the random generation and the modular exponentiation, we will count how many of these operations are required in the encryption/decryption process:

4.4.1. Proposal 0: ElGamal conventional cryptosystem

Encryption

- One random number $r_i \in \{2, q\text{-}1\}$ is generated for each message block.
- The ciphertext corresponding to each message block m_i is calculated as

  (a_i, b_i), where

  $a_i = m_i \cdot h^{r_i}$,
  $b_i = g^{r_i}$,

being $h = g^x$ the public key of the destination and *x* the secret key.

- The group $(\{a_i\}, \{b_i\})$ is sent.

Decryption

For each encrypted message block (a_i, b_i):

- The receiver calculates $h^{r_i} = {b_i}^x$.
- Then he extracts the message block $m_i = a_i \cdot ({b_i}^x)^{-1}$.

Critical operations needed:

- Encryption: one random generation, two modular exponentiations for each message block.
    - *r, h^r^, g^r^*

- Decryption: one modular exponentiation for each message block.
    - $(g^r)^x$

4.4.2. Proposal 1

In [25] a first proposal is made in order to cipher a message divided in several blocks in a more efficient way and keeping the probabilistic properties of the encryption algorithm:

Encryption

- Two random numbers r_1 and r_2 $\in$ *{2, q-1}* are generated.

- The parameters b_1, b_2 are calculated as

  $b_1 = g^{r_1}$ and $b_2 = g^{r_2}$, where *g* is the generator of the multiplicative group *Gq*.

- The ciphertext corresponding to each message block is calculated as

  $$a_i = m_i \cdot \left(h^{r_1} \oplus (h^{r_2})^{2^i}\right),$$

  where $h = g^x$ is the public key of the destination, *x* is the secret key, and $\oplus$ denotes a XOR operation.

- The group $((b_1, b_2, \{a_i\})$ is sent.

Decryption

- The receiver calculates $h^{r_1} = {b_1}^x \, mod \, p$ and $h^{r_2} = {b_2}^x \, mod \, p$.

- Then the component $\left(h^{r_1} \oplus (h^{r_2})^{2^i}\right)$ is calculated for each block.

- The receiver extracts the message block $m_i = a_i \cdot \left(h^{r_1} \oplus (h^{r_2})^{2^i}\right)^{-1}$.

Critical operations needed:

- Encryption: two random number generation and four exponentiations at first.
    - r^1, r^2, b^1, b^2, h^{r_1}, h^{r_2}

  A quadratic exponentiation is calculated for each message block:
    - $h^{r_1} \oplus (h^{r_2})^{2^i}$
- Decryption: two modular exponentiations are calculated at first.
    - h^{r_1}, h^{r_2}

  One quadratic exponentiation is needed to decrypt each message block.
    - $h^{r_1} \oplus (h^{r_2})^{2^i}$.

The idea of performing a fixed number of operations before the block encryption and reducing the operations for each block can enhance the performance of the encryption for multi-block messages. Let's see some discussion of this proposal:

4.4.2.1. *Discussion of the Proposal 1*

In [26] a cryptanalysis of the proposal 1 is made, and some issues are pointed out.

First of all, it is important to comment that, with the proposed syntax, there are some h^{r_1} and h^{r_2} values for which the $\oplus$ operation generate values that would prevent the message from being recovered from the ciphertext (for example, those ones that make the ciphertext to be 0).

r_1 and r_2 should be generated carefully to ensure they do not produce one of these critical values, and therefore the randomness factor would be noticeably reduced.

It is also demonstrated in [26] that with the solution presented in the proposal 1 the group of possible ciphertexts is smaller than the group in *Gq* since quadratic exponents are used, resulting in a weaker cryptosystem. In order to solve this problem a modification is proposed where

$a_i = m_i \cdot \left(h^{r_1} \oplus (h^{r_2})^{2^i}\right)$ changes to $a_i = m_i \cdot \left(h^{r_1} \oplus (h^{r_2})^{i}\right)$,

but no solution is proposed for the first problem.

4.4.3. Proposal 2

In [27] an improvement on the proposal 1 is presented.

Encryption

- Generate a random number *r* ∈ *{2, q-1}*.

- Each block m_i is ciphered as: $a_i = m_i \cdot (h^r + H(h^r, i))$, where *H* is a hash function.

- The group $(g^r, \{a_i\})$ is sent.

Decryption

- Receive $(g^r, \{a_i\})$.

- Calculate $h^r = (g^r)^x$ with the private key *x*.

- Knowing the index *i* of each block, reconstruct the hash function $H(h^r, i)$.

- Decrypt the message block as $m_i = a_i \cdot (h^r + H(h^r, i))^{-1}$.

Note that the algorithm can fail if $h^r + H(h^r, i) = 0$. This is verified in the encryption process, and, if it happens, a new *r* must be generated and the encryption starts from the beginning with this new value.

Critical operations needed:

- Encryption: one random number generation and two exponentiations at first.
    - g^r, h^r

  A Hash function is calculated for each message block.
    - $H(h^r, i)$

- Decryption: one modular exponentiation is calculated at first.
    - $h^r = (g^r)^x$

  One Hash function is needed to decrypt each message block.
    - $H(h^r, i)$

This algorithm is more efficient that the previous one, since a random number *r* and the exponentiation h^r have to be calculated once. Then, for each block to cipher, just a Hash function calculation is needed. Moreover, the quantity of information that has to be sent to the receiver is significantly reduced, because the size of the ciphertext is the size of the plaintext plus the factor g^r, instead of doubling the plaintext size.

4.4.3.1. Discussion of the Proposal 2

From the threat analysis point of view, the security of the cryptosystem relies on the complexity of finding, for a known ciphertext a_i and a message block m_i, the value of the parameter *w* that makes that

$$a_i = m_i \cdot (w + H(w, i))$$

It is also clear that the obtainment of one or more ciphertext-plaintext pairs of message blocks does not allow to an attacker to guess the ciphertext-plaintext values of the reminding blocks:

From the obtainment of the cryptograms $\{a_i \ldots a_n\}$ correspondent to the blocks of one message, and the obtainment of the plaintext blocks $\{m_i \ldots m_{j-1}, m_{j+1} \ldots m_n\}$ (all but m_j), it is impossible to find the block m_j from its cryptogram a_j, since an attacker can find the value of the encryption factor multiplied by the plaintext, *e_i*, from the previous information

$$e_i = \frac{a_i}{m_i} = h^r + H(h^r, i) \text{ for } i \neq j,$$

but $e_j = h^r + H(h^r, j)$ can not be inferred from them.

It seems that this proposal could be suitable for our purposes. The problem found in this last algorithm is that it loses the homomorphism characteristic of conventional ElGamal cryptosystems. That is to say that the operation of two different ciphered messages is not equal to the cipher of two operated messages (or message blocks), i.e. $E(m_A) \Theta E(m_B) = E(m_A \Phi m_B)$. Without this property, the *Integrity Proofs* can not be calculated and used to detect any message manipulation.

4.4.4. Proposal 3 (our first proposal)

To solve the problem mentioned above a new proposal has been done in order to recover the homomorphic properties of ElGamal cryptosystem based on the proposal 2.

Encryption

- Generate a random number *r Є {2, q-1}*.
- Each block m_i is encrypted as:

  $a_i = m_i \cdot (h^r \cdot H(h^r, i))$, where *H* is a hash function.
- The group $(g^r, \{a_i\})$ is sent.

Decryption

- The receiver receives $(g^r, \{a_i\})$
- Calculate $h^r = (g^r)^x$ with the private key *x*.
- Knowing the index *i* of each block, reconstruct the hash function $H(h^r, i)$.
- Decrypt the message block as $m_i = a_i \cdot (g^{r \cdot x} \cdot H(g^{r \cdot x}, i))^{-1}$.

That way the homomorphism is preserved.

Critical operations needed:

- Encryption: one random number generation and two exponentiations at first.
  - g^r, h^r

  A Hash function is calculated for each message block.

- $H(h^r, i)$

- Decryption: one modular exponentiation is calculated at first.
  - $h^r = (g^r)^x$

  One Hash function is needed to decrypt each message block.
  - $H(h^r, i)$

How could we use the Integrity Proofs?

For example, a possible *Input Integrity Proof* based on the multiplication of *n* cryptograms a_i belonging to the same message could be:

$$\prod_{i=1}^{n} a_i = \prod_{i=1}^{n} m_i \cdot h^{r \cdot n} \cdot \prod_{i=1}^{n} H(h^r, i).$$

Then, to decrypt the *Integrity Proof*, the product of all the hashes and the factor $h^{r \cdot n}$ could be calculated as

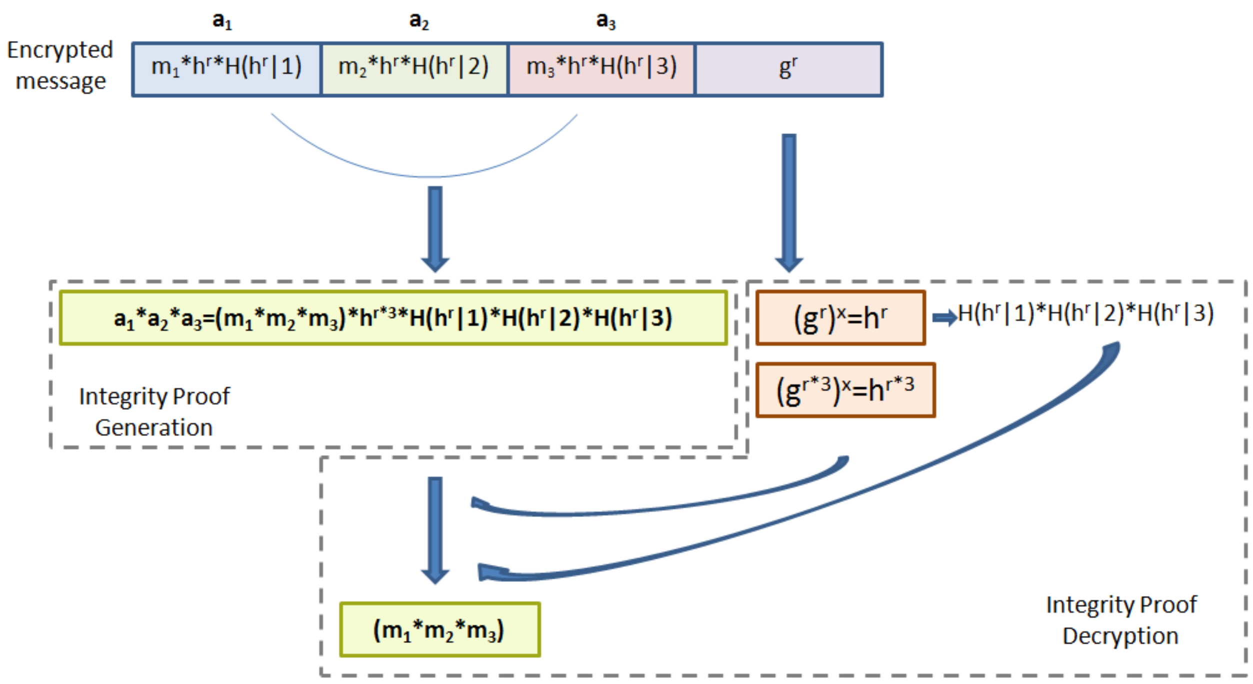
$$(g^r)^x = h^r \rightarrow H(h^r, i)$$
$$\prod_{i=1}^{n} g^r = g^{r \cdot n} \rightarrow (g^{rn})^x = h^{rn}$$

and obtain $\prod_{i=1}^{n} m_i$, so we have multiplicative homomorphism.

The *Output Integrity Proof* calculated from the decrypted message blocks would be:

$$\prod_{i=1}^{n} m_i,$$

Then, both proofs could be compared. This operation could be formalised as the operation of the message blocks of all the input and output messages.



a_1
a_2
a_3
Encrypted message
$m_1*h^r*H(h^r|1)$
$m_2*h^r*H(h^r|2)$
$m_3*h^r*H(h^r|3)$
g^r
$a_1*a_2*a_3=(m_1*m_2*m_3)*h^{r*3}*H(h^r|1)*H(h^r|2)*H(h^r|3)$
Integrity Proof Generation
$(g^r)^x=h^r$
$H(h^r|1)*H(h^r|2)*H(h^r|3)$
$(g^{r*3})^x=h^{r*3}$
$(m_1*m_2*m_3)$
Integrity Proof Decryption

Fig. 4 - 2 *Integrity Proofs* in Proposal 3

4.4.4.1. Discussion of the Proposal 3

In this solution a hash function must be generated for each block message. h^r is maintained as a constant for the blocks belonging to the same message, and the hash function gives the needed randomness component. Each time a new message has to be encrypted, a new h^r (new random *r*) is calculated to prevent the occasional repetition of the factor $h^r \cdot H(h^r, i)$ in the encryptions.

Like in the previous algorithm, the security of the cryptosystem relies on the difficulty of finding, for a known ciphertext a_i and message block m_i, the value of the parameter *w* such that

$$a_i = m_i \cdot w \cdot H(w, i)$$

In the same way, if an attacker has access to the cryptograms of the blocks of one message: $\{a_i \dots a_n\}$, and to all the decrypted blocks minus one: $\{m_i \dots m_{j-1}, m_{j+1} \dots m_n\}$, it is impossible to find the block m_j from its cryptogram a_j, due to the fact that from the previous information an attacker can find the component

$h^r \cdot H(h^r, i)$ for $i \neq j$,

but $h^r \cdot H(h^r, j)$ can not be calculated from it.

Therefore, this proposal provides a method to encrypt several blocks of one message in an efficient way without losing the homomorphic properties of the ElGamal cryptosystem. Comparing this proposal with the conventional method of independent message block codification, both efficiency in performance and size have increased.

In the following figures all the cryptographic operations have been performed using keys of 1024 bits.

4.4.4.1.2. *Performance of the Proposal 3*

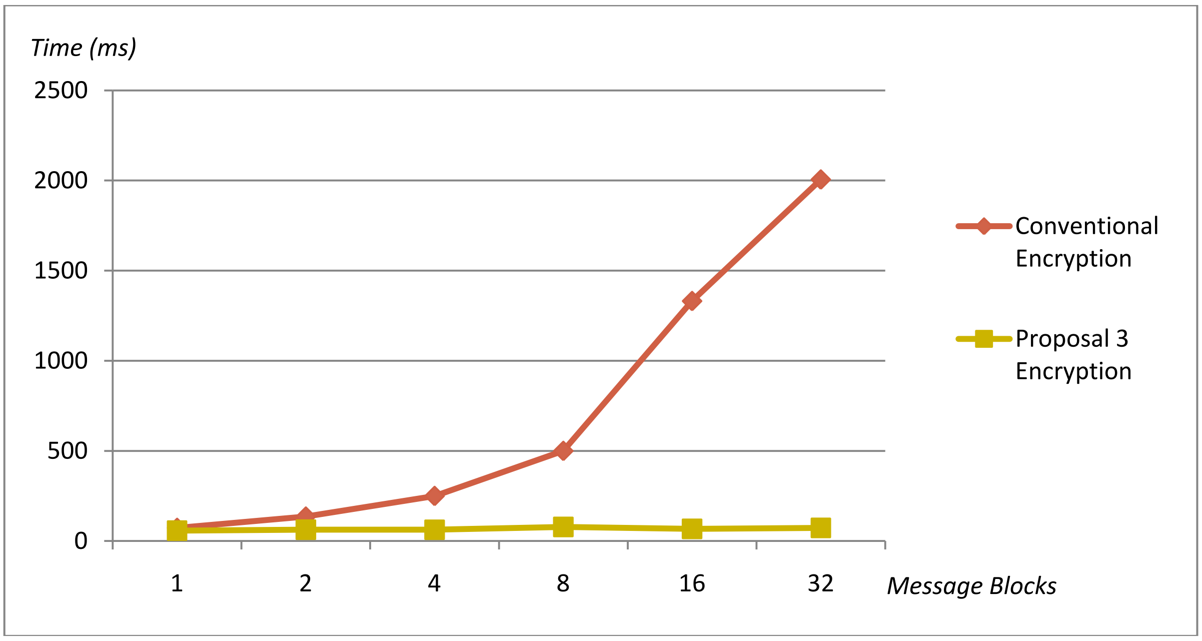


Fig. 4 - 3 Encryption time

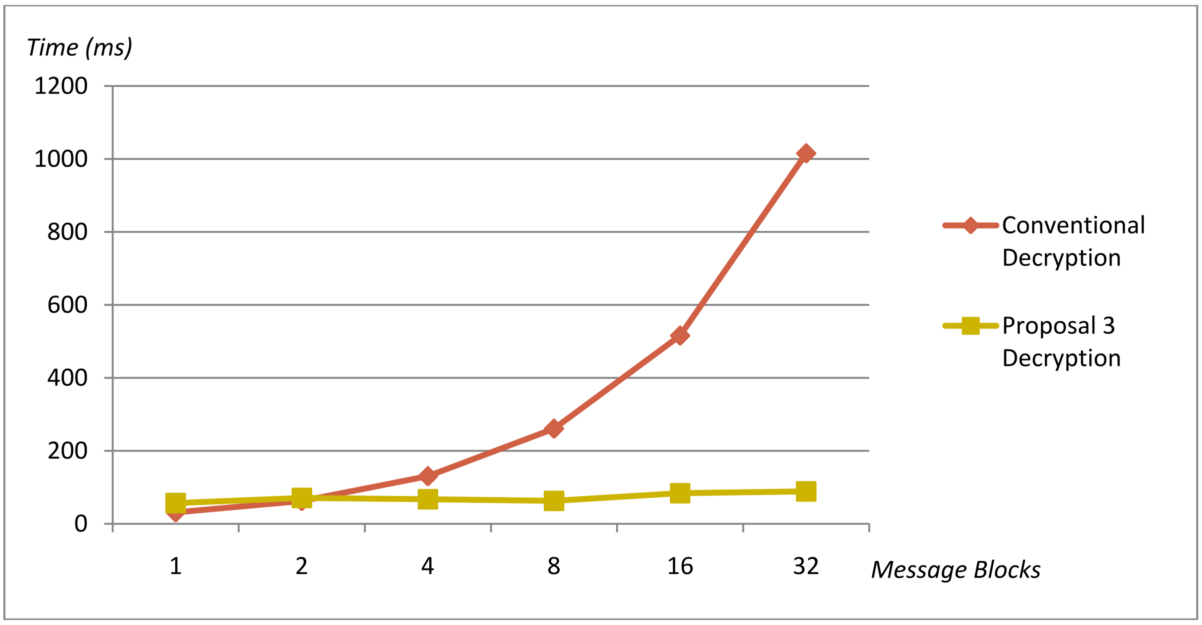


Fig. 4 - 4 Decryption time

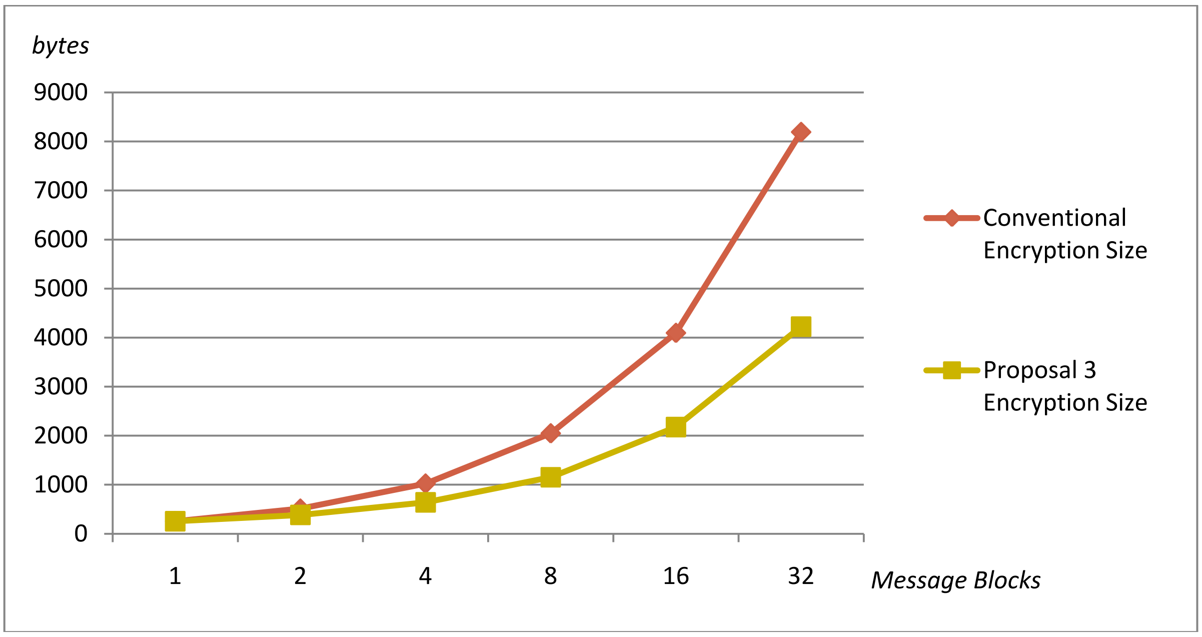


Fig. 4 - 5 Encryption size

These graphics show the time needed to encrypt and decrypt a multi-block message, comparing the performance of the conventional ElGamal cryptosystem and the design of the Proposal 3.

We can see that for short messages (few blocks) the performance of both cryptosystems is similar or even the Proposal 3 is worse, due that the fixed number of operations for each message is higher in the Proposal 3 than in the conventional cryptosystem.

When the number of blocks of a message increases, the time needed to encrypt/decrypt in the Proposal 3 is almost constant, since the variable operations (the operations for each block) in this proposal are less computationally costly than in the conventional cryptosystem.

A resume of the fixed and variable operations for each cryptosystem is presented in Table 1:

	Fixed (for each message)	**Variable (for each message block)**
Conventional Encryption		1 random generation 2 modular exponentiations
Proposal 3 Encryption	1 random generation 2 modular exponentiations	1 hash calculation
Conventional Decryption		1 modular exponentiation 1 inversion
Proposal 3 Decryption	1 modular exponentiation 1 inversion	1 hash calculation 1 inversion

Table 1

Finally, we notice that the size of encrypted messages increases faster in the case of the conventional encryption than in the Proposal 3, since while in the conventional encryption each message block doubles its size when it is encrypted, in the Proposal 3 only the one block of each message doubles its size and the others maintain their size. In the figure 4.6 the sizes of the encrypted messages using both cryptosystems can be compared:

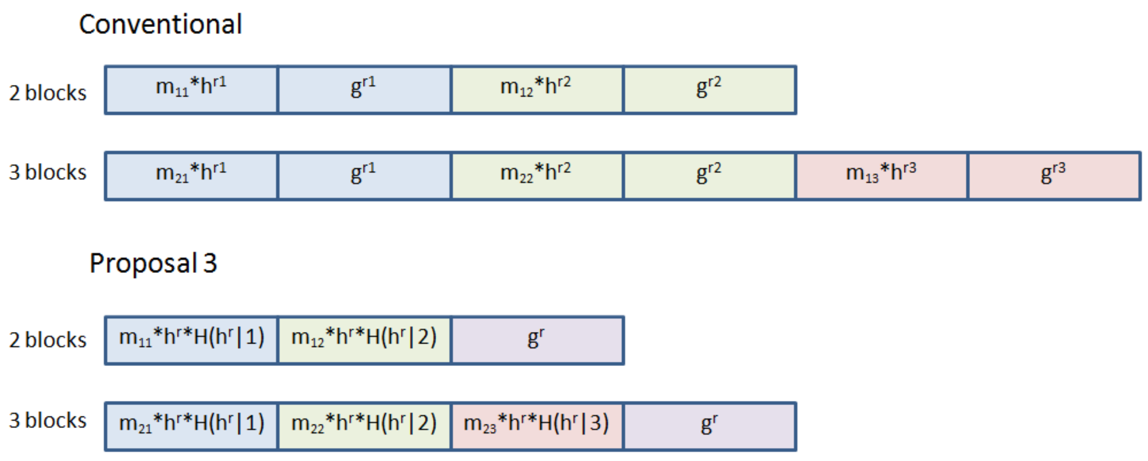


Fig. 4 - 6 Comparison of encryption sizes

We have more requirements to be accomplished by the new designed cryptosystem, for example to be able to implement re-encryption support. Therefore, the search for a suitable design must continue.

4.4.5. The re-encryption process

In re-encryption mixnets the input messages are permuted and re-encrypted. Therefore, the encrypted messages are encrypted again with new randomization values. This creates new values in the output that can not be connected to the values in the input. Using the properties of some homomorphic probabilistic algorithms such as ElGamal or Pallier, the messages are re-encrypted with the same public key *h* as in the first encryption step, but using different random numbers. Therefore, they can be decrypted in one step at the end of the mixing process with the private key *x*.

The re-encryption can be expressed as an operation on the input encrypted message where:

Being *C* the encryption of a message *m*

$$C = (m \cdot h^r, g^r)$$

The re-encryption result is

$$C' = C \cdot (h^{r'}, g^{r'}) = (m \cdot h^{r+r'}, g^{r+r'}).$$

The re-encrypted message *C'* can be decrypted in one step since

$$(g^{r+r'})^x = h^{r+r'} \text{ and } m = (m \cdot h^{r+r'}, g^{r+r'}) \cdot (g^{r+r'})^{-x}$$

If we analize this method in Proposal 3, we conclude that it is impossible to implement a re-encryption process on it, since the encryption of the message blocks is $(\{a_i \dots a_n\}, g^r)$, where $a_i = m_i \cdot (h^r \cdot H(h^r, i))$ contains the factor h^r into the hash function that codifies each message block.

Then, when the re-encryption process is done, the values $g^r$and $h^r$are updated with new randomization values to $g^{r+r'+\cdots}$ and $h^{r+r'+\cdots}$. Therefore, the factor h^r inside the Hash cannot be recovered from $(g^{r+r'+\cdots})^x$ to reconstruct this encryption component, thus the messages are not decrypted.

We need another method that randomizes the encryption of each message block, is only recoverable by the receiver and that supports the re-encryption process.

4.4.6. Proposal 4 (our second proposal)

In a first approach proposed we try encoding each message block as

Encryption

- Calculate a random integer *r*.
- Encrypt each message block m_i as

  $$a_i = m_i \cdot (h^r \cdot H(m_{i-1}, i))$$

- The encrypted message is

  $$(\{a_i \dots a_n\}, g^r)$$

Thus, the hash component is different and random at each message block (since we want to prevent the encryption of two equal message blocks to result in two equal ciphertexts (probabilistic encryption)), and there are no problems reconstructing the hash function when a re-encryption process is done, since the hash does not contain any element that could change during the re-encryption process.

Re-encryption

- Calculate a random integer *r'*.
- Re-encrypt the message $C = (\{a_i \dots a_n\}, g^r)$ as

  $$C' = C \cdot \left(h^{r'}, g^{r'}\right) = (\{a_i \dots a_n\} \cdot h^{r'}, g^{r+r'})$$

Decryption

- Calculate the factor $(g^{r+r\prime})^x = h^{r+r\prime}$.
- Calculate a hash function for each block from the recovered previous message block:

  $$H(m_{i-1}, i)$$

- Recover each message bloc m_i as

  $$m_i = a_i \cdot (h^{r+r\prime} \cdot H(m_{i-1}, i))^{-1}$$

However, we found a problem when encrypting the first message block, since we do not have a previous message block to put inside the hash function. Since it has to be recoverable by the receiver, the alternatives are

$a_1 = m_1 \cdot h^r$, or
$a_1 = m_1 \cdot h^r \cdot H(1)$

In both cases, if an attacker obtains the pair a_1, m_1, the factor h^r can be calculated as

a_1/m_1,
or $a_1 \cdot (m_1 \cdot H(1))^{-1}$.

Using m_1 the second block can be decrypted, then m_2 could be used to decrypt the third block, and so on. Therefore, the system is not secure enough since the encryption can be broken from the knowledge of one pair ciphertext-plaintext of the message blocks.

We tried a <u>second approach</u> of this proposal that involves maintaining a hash structure as a randomization factor applied over a number that is constant for all the blocks of the same message and different for each new message, and that only the receiver may know.

The constant is encrypted and sent with the other parameters as if it was a message block, in order to be recovered by the receiver.

Encryption

- Calculate two random integers r_1, r_2.

- Encrypt each message block m_i as

  $a_i = m_i \cdot h^{r_1} \cdot H(h^{r_2}, i)$

- The encrypted message is

  $(\{a_i \dots a_n\}, g^{r_1}, h^{r_1} \cdot g^{r_2})$

Decryption

- Calculate $h^{r_1} = (g^{r_1})^x$.

- g^{r_2} can be recovered from the third tuple factor as $(h^{r_1} \cdot g^{r_2}) \cdot h^{-r_1}$.

- Calculate h^{r_2} as $(g^{r_2})^x$.

- Then compute the hash function $H(h^{r_2}, i)$ for each message block.

- Recover each message block as:

  $m_i = a_i \cdot (h^{r_1} \cdot H(h^{r_2}, i))^{-1}$

As it is commented in the Proposal 2, if an attacker knows the ciphertext and the plaintext a_j and m_j of a specific block *j*, the factor $h^{r_1} \cdot H(h^{r_2}, i)$ can be discerned. Since the parameter h^{r_2} can not be extracted from the hash function because it is a unidirectional function, $H(h^{r_2}, i)$ for $i \neq j$ cannot be calculated. Therefore, the knowledge of one block message and its ciphertext does not bring additional information that helps to break the other encryptions.

Re-encryption

The multiplication of g^{r_2} by h^{r_1} allows the re-encryption of the ciphertexts multiplying the third part of the tuple by new random values $h^{r_k'}$ at each node of the mixnet.

- Thus, if we have a multi-block message encrypted as

  $$C = (\{a_i \dots a_n\}, g^{r_1}, h^{r_1} \cdot g^{r_2}) = (a, b, d),$$

  where $a_i = m_i \cdot h^{r_1} \cdot H(h^{r_2}, i)$

- the result of the re-encryption is

  $$C' = C \cdot (\{h^{r'} \dots h^{r'}\}, g^{r'}, h^{r'}) = (\{a_i' \dots a_n'\}, g^{r_1} \cdot g^{r'}, h^{r_1} \cdot h^{r'} g^{r_2}) = (a', b', c'),$$

  where $a_i' = m_i \cdot h^{r_1 + r'} \cdot H(h^{r_2}, i)$.

Decryption (after re-encryption)

- Calculate $(b')^x = h^{r_1 + r'}$, and $g^{r_2} = d'/h^{r_1 + r'}$.
- Compute $h^{r_2} = (g^{r_2})^x$.
- The factor $p_i' = h^{r_1 + r'} \cdot H(h^{r_2}, i)$ is calculated for each message block.
- Finally, the message block $m_i = a_i/p_i'$ is recovered.

Critical operations needed:

- Encryption: two random number generation and four exponentiations are calculated at first.
    - $g^{r_1}, h^{r_1}, g^{r_2}, h^{r_2}$

  A Hash function is calculated for each message block.
    - $H(h^{r_2}, i)$

- Decryption: two modular exponentiations are calculated.
    - $h^{r_1} = (g^{r_1})^x$
    - $h^{r_2} = (g^{r_2})^x$

  One Hash function is needed to decrypt each message block.
    - $H(h^{r_2}, i)$

How could we use the Integrity Proofs?

For example, a possible *Input Integrity Proof* based on the multiplication of *n* cryptograms a_i belonging to the same message could be:

$$\prod_{i=1}^{n} a_i = \prod_{i=1}^{n} m_i \cdot h^{r_1 \cdot n} \cdot \prod_{i=1}^{n} H(h^{r_2}, i).$$

Then, to decrypt the *Integrity Proof*, the product of all the hashes and the factor $h^{r_1 \cdot n}$ could be calculated as

$$(g^{r_1})^x = h^{r_1} \rightarrow g^{r_2} = (g^{r_2} \cdot h^{r_1})/h^{r_1}$$
$$(g^{r_2})^x = h^{r_2} \rightarrow H(h^{r_2}, i)$$
$$\prod_{i=1}^{n} g^{r_1} = g^{r_1 \cdot n} \rightarrow (g^{r_1 \cdot n})^x = h^{r_1 \cdot n}$$

and obtain $\prod_{i=1}^{n} m_i$, so we have multiplicative homomorphism.

The *Output Integrity Proof* calculated from the decrypted message blocks would be:

$$\prod_{i=1}^{n} m_i$$

Then, both proofs could be compared. This operation could be generalised as the operation of the message blocks of all the input and output messages.

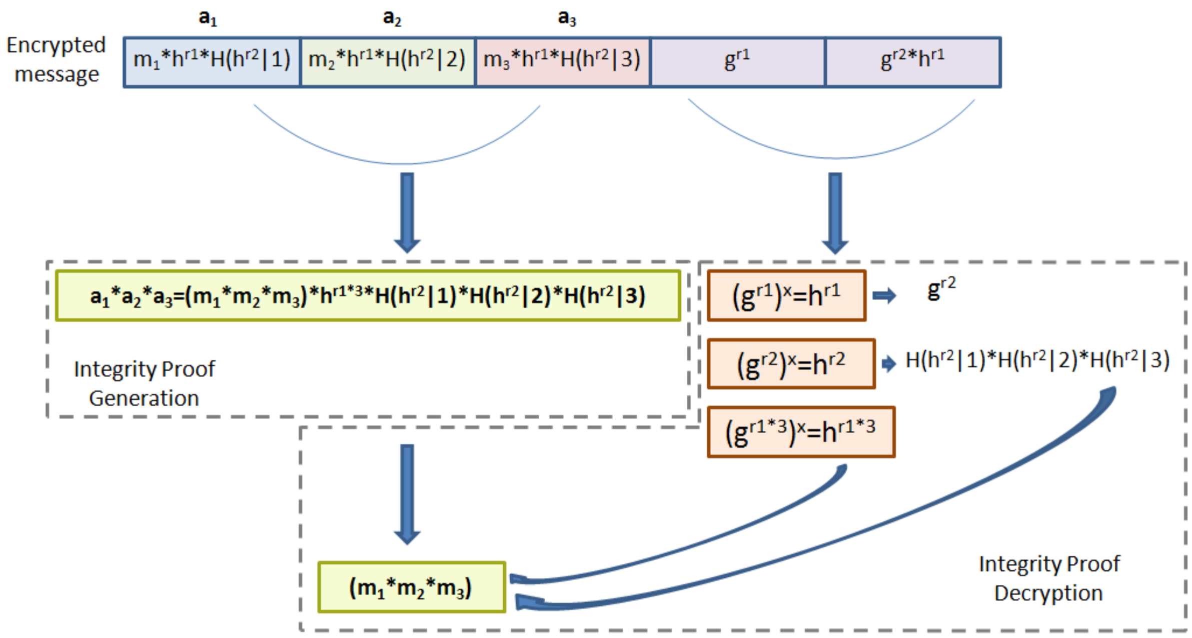


Fig. 4 - 7 *Integrity Proofs* in Proposal 4

4.4.6.1. *Performance of the Proposal 4*

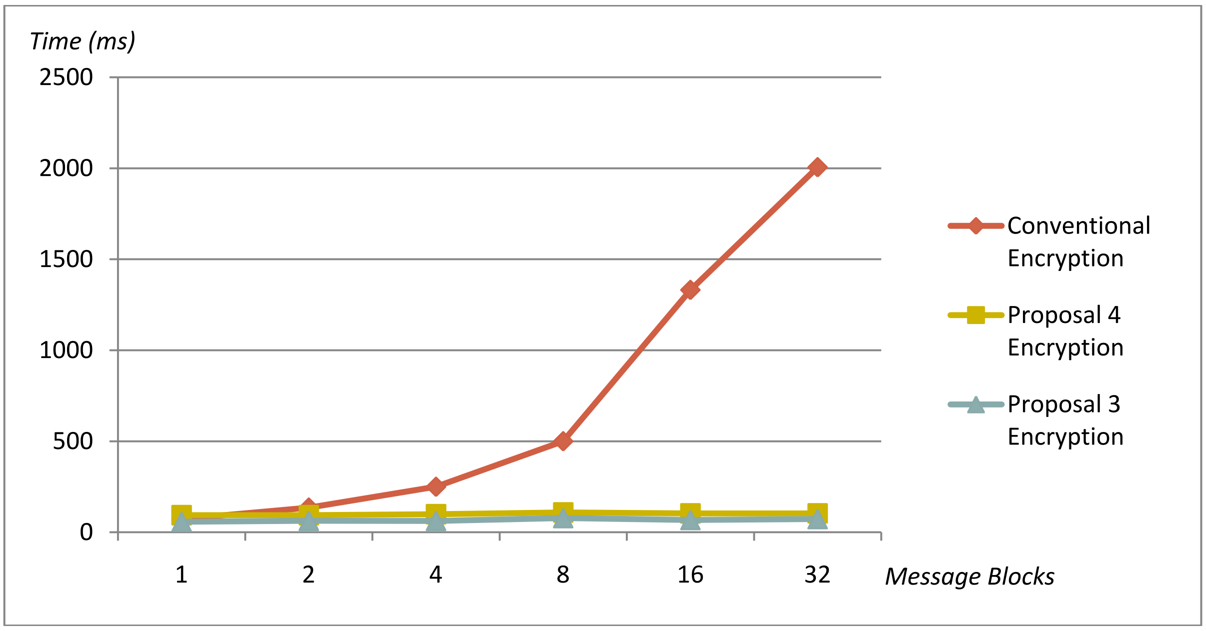


Fig. 4 - 8 Encryption time

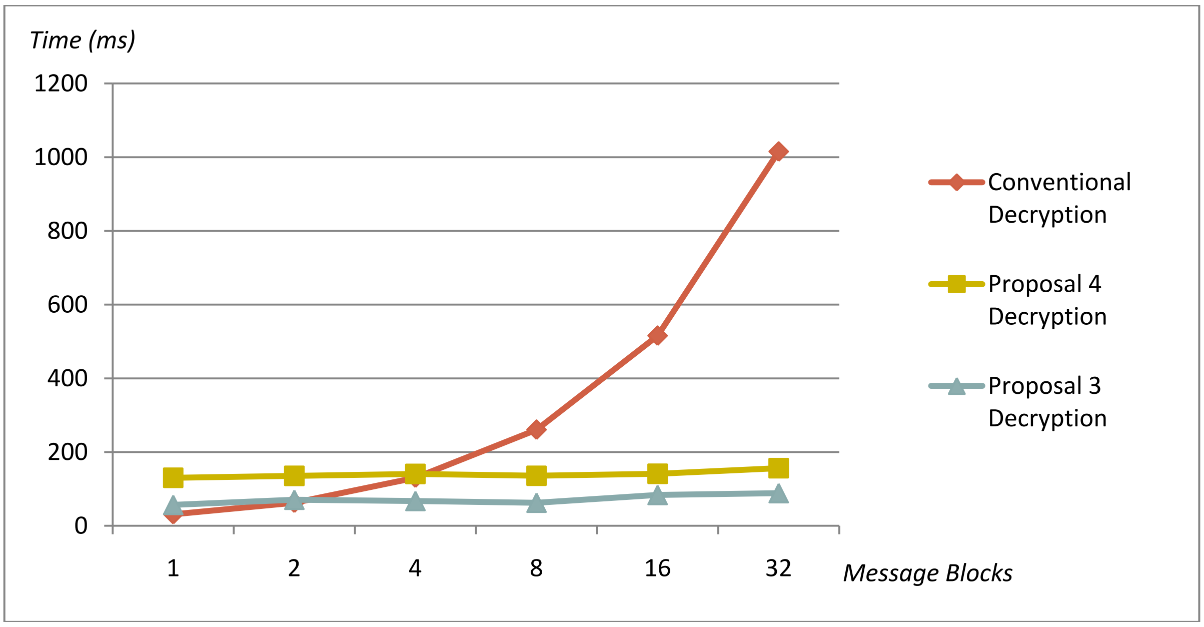


Fig. 4 - 9 Decryption time

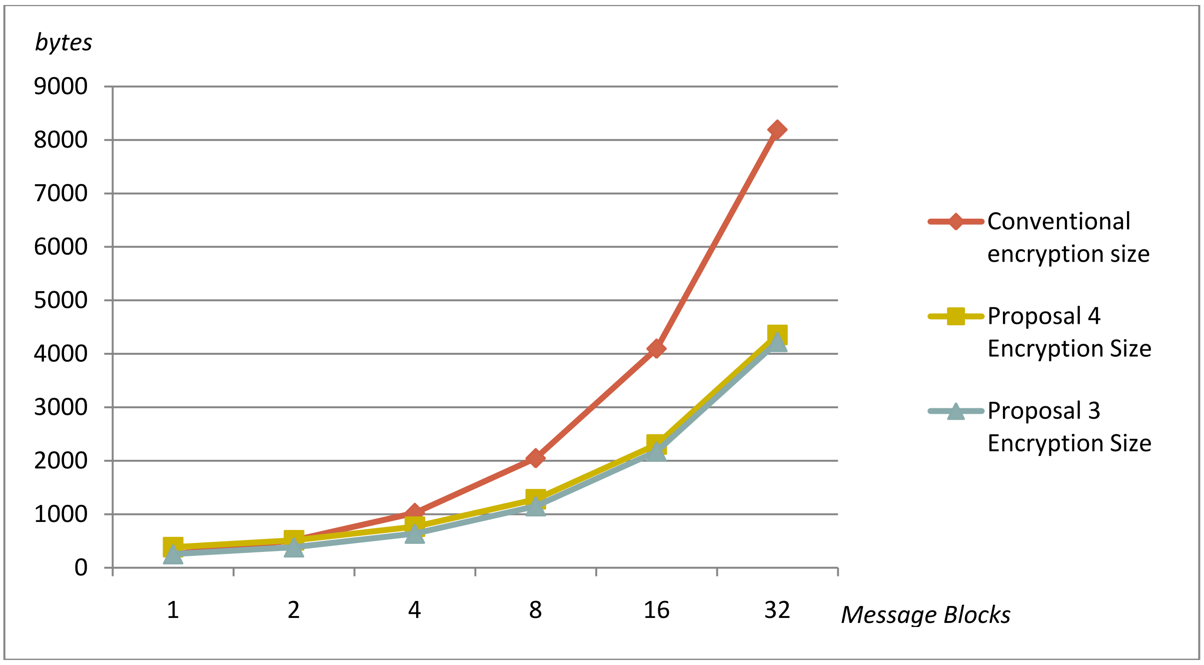


Fig. 4 - 10 Encryption size

These graphics show the time needed to encrypt and decrypt a multi-block message, comparing the performance of the conventional ElGamal cryptosystem, the designed cryptosystem of the Proposal 3, and the Proposal 4.

We can see that the performance for encryption/decryption is very similar for Proposal 3 and Proposal 4, due to that they are based on the same idea of reducing the cost of the variable (per block operations). Therefore, the time for encryption/decryption in the conventional cryptosystem increases with the message number of blocks, while in the Proposal 3 and 4 is almost constant.

The time for decryption increases from the Proposal 3 to the Proposal 4 since the decryption process is more complex in the second proposal. Even so, it is constant while in the conventional cryptosystem increases with the message size, so the performance of the Proposal 4 is still better.

A resume of the fixed and variable operations for each cryptosystem is presented in Table 2:

	Fixed (for each message)	**Variable (for each message block)**
Conventional Encryption		1 random generation 2 modular exponentiations
Proposal 4 Encryption	2 random generation 4 modular exponentiations	1 hash calculation
Conventional Decryption		1 modular exponentiation 1 inversion
Proposal 4 Decryption	2 modular exponentiations 1 inversion	1 hash calculation 1 inversion

Table 2

Finally, the encryption size increases faster in the case of the conventional encryption than in the Proposal 4, due to that while in the conventional encryption each message block doubles its size when it is encrypted, only the first block of each message doubles its size, an additional block is added and the others maintain their size in the Proposal 4. Due to this additional block, the encryption size is slightly larger in the Proposal 4 than in the Proposal 3, but the difference is negligible when the message size increases.

In the figure 4.11 there is a comparison of the size of the encrypted messages using the conventional cryptosystem and the Proposal 4 version 2. The comparison is done using 2-block and 3-block messages.

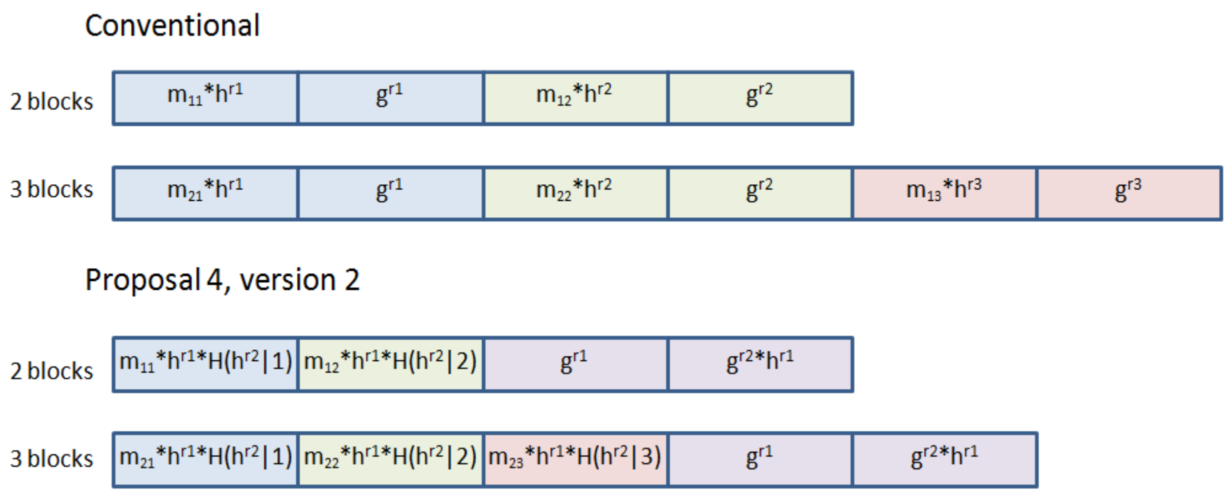


Fig. 4 - 11 Comparison of encryption sizes

4.4.7. Problems found in the previous proposals

Once we thought we had our definitive proposal to achieve an efficient encryption scheme for multi-block messages based in the ElGamal cryptosystem, we found two more problems to face at:

4.4.7.1. Message traceability through re-encryption processes

The first problem is that there is a factor in the encrypted message that remains constant through the successive re-encryptions on each node.

If we consider the encrypted message

$$C = (\{a_i \dots a_n\}, g^{r_1}, h^{r_1} \cdot g^{r_2}) = (a, b, d),$$

where $a_i = m_i \cdot h^{r_1} \cdot H(h^{r_2}, i)$,

and the reencryption result is

$$C' = C \cdot (\{h^{r'} \dots h^{r'}\}, g^{r}, h^{r'}) = (\{a_i' \dots a_n'\}, g^{r_1} \cdot g^{r'}, h^{r_1} \cdot h^{r'} g^{r_2}) = (a', b', c'),$$

where $a_i{}' = m_i \cdot h^{r_1} \cdot h^{r'} \cdot H(h^{r_2}, i)$, we can see that the factor $H(h^{r_2}, i)$ is constant through the re-encryptions.

This constant can help an attacker to guess the path of a message through the mix-nodes.

As it is shown in the figure 4.12, an attacker could calculate at each node and for each encrypted message the result of dividing two of the pieces of the ciphertext:

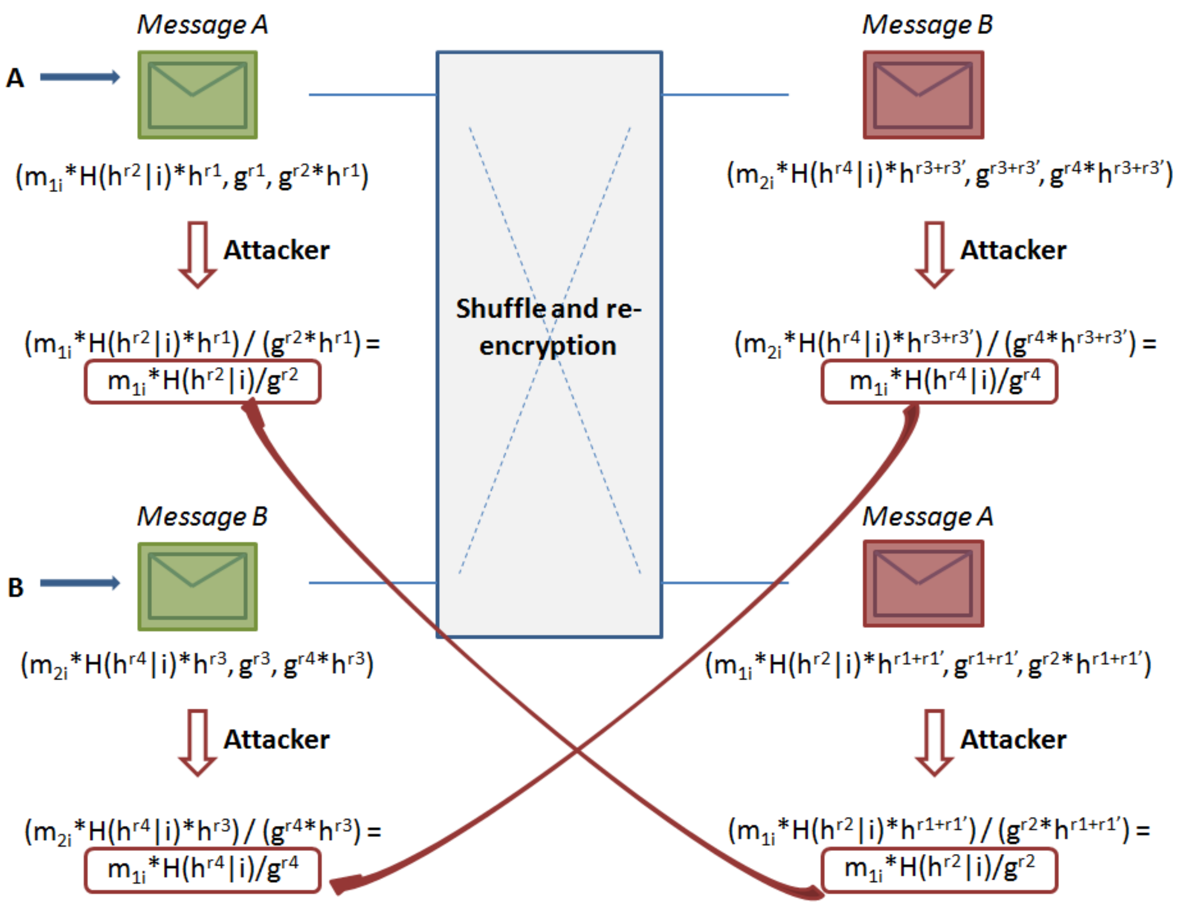

Message A
A
(m1i*H(hr2|i)*hr1, gr1, gr2*hr1)
Attacker
(m1i*H(hr2|i)*hr1) / (gr2*hr1) =
m1i*H(hr2|i)/gr2
Message B
B
(m2i*H(hr4|i)*hr3, gr3, gr4*hr3)
Attacker
(m2i*H(hr4|i)*hr3) / (gr4*hr3) =
m1i*H(hr4|i)/gr4
Shuffle and re-encryption
Message B
(m2i*H(hr4|i)*hr3+r3', gr3+r3', gr4*hr3+r3')
Attacker
(m2i*H(hr4|i)*hr3+r3') / (gr4*hr3+r3') =
m1i*H(hr4|i)/gr4
Message A
(m1i*H(hr2|i)*hr1+r1', gr1+r1', gr2*hr1+r1')
Attacker
(m1i*H(hr2|i)*hr1+r1') / (gr2*hr1+r1') =
m1i*H(hr2|i)/gr2

Fig. 4 - 12 An attacker tracks back the re-encrypted messages

Node 1:

$$\frac{a_i}{d} = \frac{m_i \cdot H(h^{r_2}, i) \cdot h^{r_1}}{g^{r_2} \cdot h^{r_1}} = \frac{m_i \cdot H(h^{r_2}, i)}{g^{r_2}}$$

Node 2:

$$\frac{a_i'}{d'} = \frac{m_i \cdot H(h^{r_2}, i) \cdot h^{r_1+r'}}{g^{r_2} \cdot h^{r_1+r'}} = \frac{m_i \cdot H(h^{r_2}, i)}{g^{r_2}}$$

Since the result of the operation is constant in both nodes, the attacker could track back the message from the decryption step to the reception, where it is related to the sender, and break the voter privacy.

4.4.7.2. The 300 attack

The second problem is an attack performed in [16] to a mixnet designed in [15]

That mixnet used an ElGamal cryptosystem and *Integrity Proofs* based on similar concepts to ours. One aspect that Wikström pointed out of that mixnet was that it was not able to detect a specific attack to the messages with its *Integrity Proofs*.

An example of calculation of the *Integrity Proofs* as the multiplication of the processed messages has been explained in the proposals 3 and 4.

Malleability

The ElGamal encryption scheme is malleable. That means that, if we multiply an integer by a ciphertext the result is the encryption of integer multiplied by the original encrypted message:

$$Encryption(m) \rightarrow c = (m \cdot h^r, g^r)$$

$$x \cdot c = (x \cdot m \cdot h^r, g^r)$$

$$Decryption(x \cdot c) = x \cdot m$$

Since this malleability is maintained in our proposals, we will explain how it can be a problem based on the second version of the Proposal 4, when the verification of the process is done through the comparison of the *Integrity Proofs*:

Integrity Proofs

- Assume two parties, A and B, that send two encrypted messages as:

$$A \rightarrow m_A$$

$$Encryption(m_A) = C_A = (\{a_{Ai} \dots a_{An}\}, g^{r_{A1}}, h^{r_{A1}} \cdot g^{r_{A2}}) = (a_A, b_A, d_A),$$

where $a_{Ai} = m_{Ai} \cdot h^{r_{A1}} \cdot H(h^{r_{A2}}, i)$

$$B \rightarrow m_B$$

$$Encryption(m_B) = C_B = (\{a_{Bi} \dots a_{Bn}\}, g^{r_{B1}}, h^{r_{B1}} \cdot g^{r_{B2}}) = (a_B, b_B, d_B),$$

where $a_{Bi} = m_{Bi} \cdot h^{r_{B1}} \cdot H(h^{r_{B2}}, i)$

- The *Input Integrity Proof* is

$$I_I = \prod_{i=1}^{n} (a_{Ai} \cdot a_{Bi}) = \prod_{i=1}^{n} (m_{Ai} \cdot m_{Bi}) \cdot (h^{r_{A1} \cdot n} \cdot h^{r_{B1} \cdot n}) \cdot \prod_{i=1}^{n} (H(h^{r_{A2}}, i) \cdot H(h^{r_{B2}}, i))$$

- The product of all the hashes and the factors $h^{r_{A1}}$ and $h^{r_{B1}}$ must be calculated to decrypt the *Input Integrity Proof*:

$(g^{r_{A1}})^x = h^{r_{A1}} \to g^{r_{A2}} \to (g^{r_{A2}})^x = h^{r_{A2}} \to H(h^{r_{A2}}, i)$ for *i=1…n.*

$(g^{r_{B1}})^x = h^{r_{B1}} \to g^{r_{B2}} \to (g^{r_{B2}})^x = h^{r_{B2}} \to H(h^{r_{B2}}, i)$ for *i=1…n.*

$\prod_{i=1}^{n}(g^{r_{A1}}) = g^{r_{A1} \cdot n} \to (g^{r_{A1} \cdot n})^x = h^{r_{A1} \cdot n}$

$\prod_{i=1}^{n}(g^{r_{B1}}) = g^{r_{B1} \cdot n} \to (g^{r_{B1} \cdot n})^x = h^{r_{B1} \cdot n}$

- Finally, we obtain

$$I_I' = \prod_{i=1}^{n}(m_{Ai} \cdot m_{Bi})$$

- The *Output Integrity Proof* is calculated from the decrypted messages, so the result is

$$I_O = \prod_{i=1}^{n}(m_{Ai} \cdot m_{Bi})$$

The comparison of these two proofs can be used to detect the possible manipulation of the messages during the mixing process.

Attack

Due to the malleability property of the cryptosystem, if someone multiplies the ciphertext by a factor (i.e. an integer), the result of the decryption is the plaintext message multiplied by this factor:

- If an attacker multiplies an integer by one ciphertext and its inverse by the other, for example:

$300 \cdot C_A$

$300^{-1} \cdot C_B$

- The result of the *Input Integrity Proof* is

$$\begin{aligned} I_I = \prod_{i=1}^{n}(a'_{Ai} \cdot a'_{Bi}) \\ &= \prod_{i=1}^{n}(300 \cdot m_{Ai} \cdot (300^{-1}) \cdot m_{Bi}) \cdot (h^{r_{A1} \cdot n} \cdot h^{r_{B1} \cdot n}) \\ &\cdot \prod_{i=1}^{n}(H(h^{r_{A2}}, i) \cdot H(h^{r_{B2}}, i)) \end{aligned}$$

- And the decrypted *Input Integrity Proof* is

$$I_I' = \prod_{i=1}^{n}(300 \cdot m_{Ai} \cdot (300^{-1}) \cdot m_{Bi}) = \prod_{i=1}^{n}(m_{Ai} \cdot m_{Bi})$$

- The result of the *Output Integrity Proof* is

$$I_O = \prod_{i=1}^{n} (300 \cdot m_{Ai} \cdot (300^{-1}) \cdot m_{Bi}) = \prod_{i=1}^{n} (m_{Ai} \cdot m_{Bi})$$

Therefore, the manipulation is not detected due to the fact that both proofs are equal (and equal to the ones without manipulation), although the decrypted messages do not match the original ones.

This is a problem that we have called *The 300 Attack*, for the value of the factor used to multiply and manipulate the messages while we were thinking about a solution for it. Our new proposal is aimed to solve these two problems, preventing *The 300 Attack* and the message tracing in the mix-nodes.

4.4.8. Proposal 5 (our third proposal)

Our solution for the problems presented above requires some reasoning steps:

4.4.8.1. Solution to message traceability through re-encryption processes

Any relationship between the elements of the encrypted messages must be avoided in order to prevent an attacker from finding a value that remains constant through re-encryptions.

For this purpose, a modification of the Proposal 4 has been done:

- Instead of creating a public key h such as $h = g^x mod\ p$, where x is the private key, two public and private keys are created using the same public parameters *p* and *g*:

  $h_1 = g^{x_1}$, and $h_2 = g^{x_2}$

- While in the Proposal 4 a message was encrypted as

  $(\{a_i \dots a_n\}, g^{r_1}, h^{r_1} \cdot g^{r_2}) = (a, b, d)$, where $a_i = m_i \cdot h^{r_1} \cdot H(h^{r_2}, i)$

  In the Proposal 5 the encryption is performed as

  $(\{a_i \dots a_n\}, g^r, h_2^r \cdot y)$,

  where $a_i = m_i \cdot h_1^r \cdot H(y, i)$ and *y* is a random number $1 \leq y \leq q - 1$.

Since we have a second public key h_2, we can encrypt a random number *y* instead of g^{r_2} using it. Then, *y* can be used to obfuscate the message-block content within the hash $H(y, i)$.

In the Proposal 4, since the factor h^{r_1} could be discovered during the decryption process, it was important to protect the obfuscating hash $H(h^{r_2}, i)$ putting inside a number that just could be calculated having the secret key ($(g^{r_2})^x = h^{r_2}$) in order to preserve the secrecy of the encrypted votes received before the mixing process.

Now, due that the two public keys are unrelated (from the point of view that it is very difficult to know of one of them exponentiated to a random value from the knowledge of the other exponentiated to the same value), we can use directly the random number *y* instead of g^{r_2}, and then decrease the number of modular exponentiations required in the encryption/decryption process.

If an attacker tries to find a factor that remains constant through the different re-encryption processes calculating $\frac{a}{c}$ or $\frac{a}{c\cdot d}$ the result always depends on the factor $(g^r)^{x_1 \pm x_2}$, where *r* is the randomization factor in the encryption. Since the value of *r* changes at each re-encryption, the value of $(g^r)^{x_1 \pm x_2}$ changes too, and no constant factors can be found.

4.4.8.2. *Solution to undetected message manipulation*

Now that the first problem has been solved, we should find a method to detect if there has been any manipulation of the messages that has not been detected through the *Integrity Proofs*.

Given that we are not able to detect some attacks, as for example *The 300 Attack*, using group verifications as the *Integrity Proofs*, we are forced to check one by one the integrity of all the messages in the decryption step.

If the mixing verification process was enhanced, perhaps there would not be a need for these individual verifications, but we think it is still useful to be able to verify the integrity of each individual message if it is desired.

In case a manipulation was detected, the mixing process could be repeated to ensure its correctness. Therefore, during this verification process both the correct and incorrect messages must not be discovered as plaintexts. Otherwise we could reveal too much information before repeating the mixing, uncovering the paths of the shuffled messages and breaking the voter privacy.

The use of a double encryption could be a possible solution: in this scheme the upper layer encrypts the lower message encryption layer jointly with its hash function:

$$E_1(E_2(m),H(E_2(m)))$$

When the first layer is decrypted, the message integrity can be verified before decrypting the second layer, checking the connection $E_2(m) \rightarrow H(E_2(m))$. If this verification fails, the message manipulation has been detected without revealing any plaintext message.

This procedure has been followed in [48] and in [15].

Although a similar system is required in the cryptosystem to be designed since *Integrity Proofs* do not detect all the possible manipulations, the double encryption is avoided in order to achieve a lower computational cost.

Combining these two ideas we have designed a new proposal from the modification of the Proposal 4 in order to resolve the problems explained above.

Key Generation

- Choose two random integers x_1 and x_2 where $1 \leq x_1, x_2 \leq q-2$ as the secret keys.
- Calculate two public keys $h_1 = g^{x_1}$ and $h_2 = g^{x_2}$.

It is assumed that the entity in charge of vote decryption has both keys x_1 and x_2, although they could be given to different organizations in order to distribute the trust.

Encryption

- Generate two random numbers, *r* and *y*, such that $2 \leq r, y \leq q-2$.
- Divide the message *m* in different message blocks m_i such that $[m_i+s_H] < q$, where s_H is the size of a hash function that will be attached to the message block (i.e. if SHA1 is used the length is 160 bits).
- Encrypt the message *m* as

  $(\{a_i, \dots, a_n\}, g^r, {h_2}^r \cdot y)$,

  where $a_i = (m_i \cdot H(y,i) || H(m_i \cdot H(y,i))) \cdot {h_1}^r$
- Send $(a,b,d) = (\{a_i, \dots, a_n\}, g^r, {h_2}^r \cdot y)$

Re-encryption

- Generate a random number, *r'*, such that $2 \leq r' \leq q-2$.
- Thus, if we have a multi-block message encrypted as

$$C = (\{a_i, \dots, a_n\}, g^r, {h_2}^r \cdot y) = (a,b,d),$$

- The re-encryption result is

$$C' = C \cdot (\left\{{h_1}^{r'} \dots {h_1}^{r'}\right\}, g^{r'}, {h_2}^{r'}\Big) = \Big(\{a_i' \dots a_n'\}, g^r \cdot g^{r'}, {h_2}^r \cdot {h_2}^{r'} \cdot y\Big),$$

$$\text{where } a_i' = (m_i \cdot H(y,i) || H(m_i \cdot H(y,i))) \cdot {h_1}^{r+r'} = (a', b', d').$$

Partial Decryption

- The receiver calculates ${h_1}^{r+r\prime} = (g^{r+r\prime})^{x_1}$.
- Then obtains

  $m_i \cdot H(y,i) || H(m_i \cdot H(y,i)) = \frac{a_i\prime}{{h_1}^{r+r\prime}}$

  for each message block.

Hash Verification

Before keeping on with the decryption process, a verification process must be done at each individual message in order to detect attacks like the *The 300 Attack* using the component

$$m_i \cdot H(y,i)||H\left(m_i \cdot H(y,i)\right),$$

for each enrypted message or message block:

- The verifier divides it in two pieces:

$$s = m_i \cdot H(y,i)$$

$$t = H\left(m_i \cdot H(y,i)\right)$$

- Then checks that *H(s)=t*.

If the verification process succeeds, the process can continue. Otherwise it is stopped and an error is reported.

Final Decryption

- The receiver calculates ${h_2}^{r+r'} = (g^{r+r'})^{x_2}$,
- Obtains $y = d'/{h_2}^{r+r'}$
- Then calculates the hash *H(y|i)* for each message block,
- And recovers each one as $m_i = [m_i \cdot H(y,i)] \cdot H(y,i)^{-1}$.

A message divided in two blocks and then encrypted looks like this:

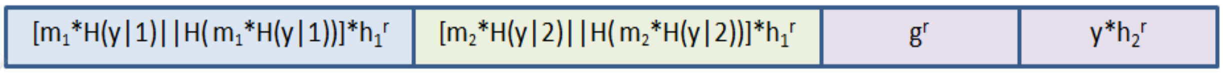

$[m_1*H(y|1)||H(m_1*H(y|1))]*h_1^r$
$[m_2*H(y|2)||H(m_2*H(y|2))]*h_1^r$
g^r
$y*h_2^r$

Fig. 4 - 13 Message encrypted using the Proposal 5

Critical operations needed:

- Encryption: two random number generation and three exponentiations at first.
  - $g^r, {h_1}^r, {h_2}^r, y$

  Two Hash functions are calculated for each message block.
  - $H(y,i), H\left(m_i \cdot H(y,i)\right)$
- Decryption: two modular exponentiations are calculated.
  - ${h_1}^r = (g^r)^{x_1}$
  - ${h_2}^r = (g^r)^{x_2}$

One Hash function is needed to decrypt each message block.

- $H(y, i)$

- Verification: a hash function is needed in order to verify the integrity of each message block.
  - $H\big(m_i \cdot H(y, i)\big)$

How could we use the Integrity Proofs?

One of the scopes of the new cryptosystem is to be able to calculate *Integrity Proofs* before and after a particular process (i.e. a mixing process that re-encrypts and decrypts the input messages) that can be compared in order to detect unauthorized modifications in the set of input messages.

An example of how these *Integrity Proofs* can be calculated and compared in this Proposal 5 is presented here:

<u>Input Integrity Proof Generation</u>

- From a set of messages in the process (mixing) input, each one being encrypted as

  $(\{a_i, \dots, a_n\}, g^r, {h_2}^r \cdot y),$

  where $a_i = (m_i \cdot H(y, i)||H\big(m_i \cdot H(y, i)\big)) \cdot {h_1}^r$

- An *Input Integrity Proof* is calculated as

$$I_I = \left(\prod_{j=1}^{k}\prod_{i=1}^{n} a_{ij}, \prod_{j=1}^{k}\prod_{i=1}^{n} g^{r_j}\right)$$
$$= \left(\prod_{j=1}^{k}\prod_{i=1}^{n} (m_{ij} \cdot H(y_j, i)||H\left(m_{ij} \cdot H(y_j, i)\right)) \cdot {h_1}^{r_j}, \prod_{j=1}^{k}\prod_{i=1}^{n} g^{r_j}\right)$$

  Where *j=1...k*, being *k* the number of input messages and *i=1…n*, being *n* the number of blocks of each input message.

This proof is calculated based on all the blocks of the set of messages in the process (mixing) input (we assume that all the encrypted messages have the same size).

<u>Output Integrity Proof Generation</u>

- From the set of messages in the process (mixing) output, where each message block is partially decrypted as

$$m_i \cdot H(y, i)||H\big(m_i \cdot H(y, i)\big)$$

- The *Output Integrity Proof* can be calculated as

$$I_O = \left(\prod_{j=1}^{k} \prod_{i=1}^{n} [m_{ij} \cdot H(y_j, i)) || H\left(m_{ij} \cdot H(y_j, i)\right)] \right)$$

Where *j=1...k*, being *k* the number of output messages, and *i=1…n*, being *n* the number of blocks of each output message.

This proof is calculated based on all the blocks of all the messages in the process (mixing) output.

Input Integrity Proof Decryption

In order to compare the *Input IntegrityProof* and the *Output Integrity Proof* in a process where the output messages are decrypted or partially decrypted, the *Input Integrity Proof* has to be decrypted too:

- From the *Input Integrity Proof*

$$I_I = \left(\prod_{j=1}^{k} \prod_{i=1}^{n} a_{ij}, \prod_{j=1}^{k} \prod_{i=1}^{n} g^{r_j} \right)$$
$$= \left(\prod_{j=1}^{k} \prod_{i=1}^{n} (m_{ij} \cdot H(y_j, i) || H\left(m_{ij} \cdot H(y_j, i)\right)) \cdot {h_1}^{r_j}, \prod_{j=1}^{k} \prod_{i=1}^{n} g^{r_j} \right)$$

the receiver calculates

$$\prod_{j=1}^{k} \prod_{i=1}^{n} ({h_1}^{r_j}) = \prod_{j=1}^{k} ({h_1}^{r_j \cdot n}) = \left(\prod_{j=1}^{k} \prod_{i=1}^{n} (g^{r_j}) \right)^{x_1} = \left(\prod_{j=1}^{k} (g^{r_j \cdot n}) \right)^{x_1}$$

- Then obtains the decryption of the *Input Integrity Proof* as

$$I_I' = \left(\prod_{j=1}^{k} \prod_{i=1}^{n} [m_{ij} \cdot H(y_j, i) || H\left(m_{ij} \cdot H(y_j, i)\right)] \right) = I_I \cdot \left(\prod_{j=1}^{k} ({h_1}^{r_j \cdot n}) \right)^{-1}$$

Integrity Proofs Verification

The Output Integrity Proof I_O and the Decrypted Input Integrity Proof I_I' are compared. If the messages have been manipulated during the process (mixing), these two proofs do not match. In case of *The 300 attack* the manipulation is not detected until the Hash verification step.

Procedure

Using a case study where the encrypted messages are encrypted votes and the process that makes a transformation over the encrypted messages is a Mixnet which does a re-encryption process at each mix-node and a decryption one in the last node, the steps to follow in order to take profit of the main advantages of this encryption proposal are:

1. Message Encryption
2. Input Integrity Proof Generation
3. Message Re-encryption
4. Intermediate Integrity Proof Generation
5. Partial Decryption
6. Output Integrity Proof Generation
7. Input and Intermediate Integrity Proof Decryption
8. Integrity Proofs Verification
9. Hash Verification
10. Final Decryption

In the figures 4.14, 4.15, 4.16 and 4.17 an example of the mixing procedure for two messages is described.

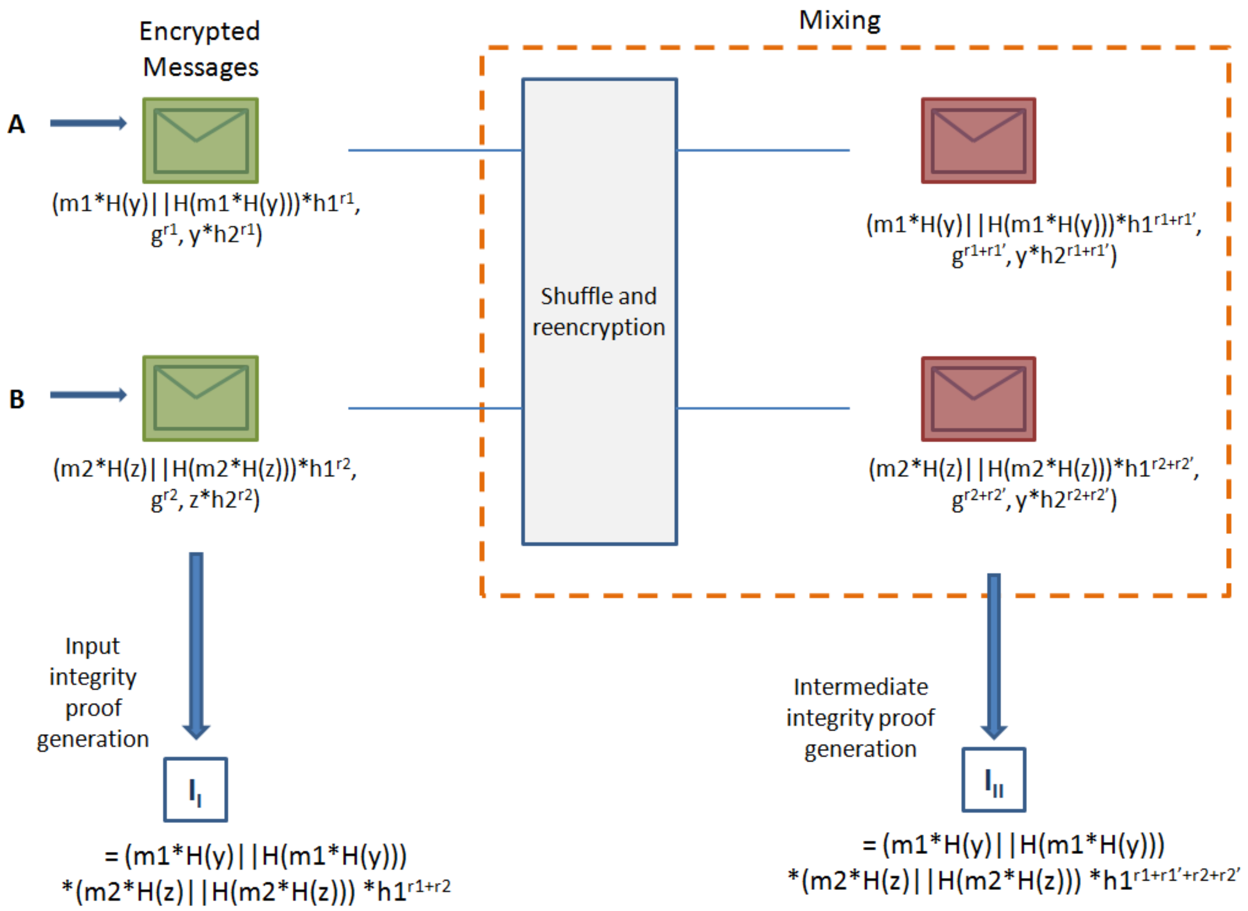

Encrypted Messages
Mixing
A
B
(m1*H(y)||H(m1*H(y)))*h1^r1, g^r1, y*h2^r1)
(m2*H(z)||H(m2*H(z)))*h1^r2, g^r2, z*h2^r2)
Shuffle and reencryption
(m1*H(y)||H(m1*H(y)))*h1^r1+r1', g^r1+r1', y*h2^r1+r1')
(m2*H(z)||H(m2*H(z)))*h1^r2+r2', g^r2+r2', y*h2^r2+r2')
Input integrity proof generation
I_I
= (m1*H(y)||H(m1*H(y))) *(m2*H(z)||H(m2*H(z))) *h1^r1+r2
Intermediate integrity proof generation
I_II
= (m1*H(y)||H(m1*H(y))) *(m2*H(z)||H(m2*H(z))) *h1^r1+r1'+r2+r2'

Fig. 4 - 14 Message encryption. First shuffle and re-encryption process in the mixnet. Calculation of *Input* and *Intermediate Integrity Proofs*

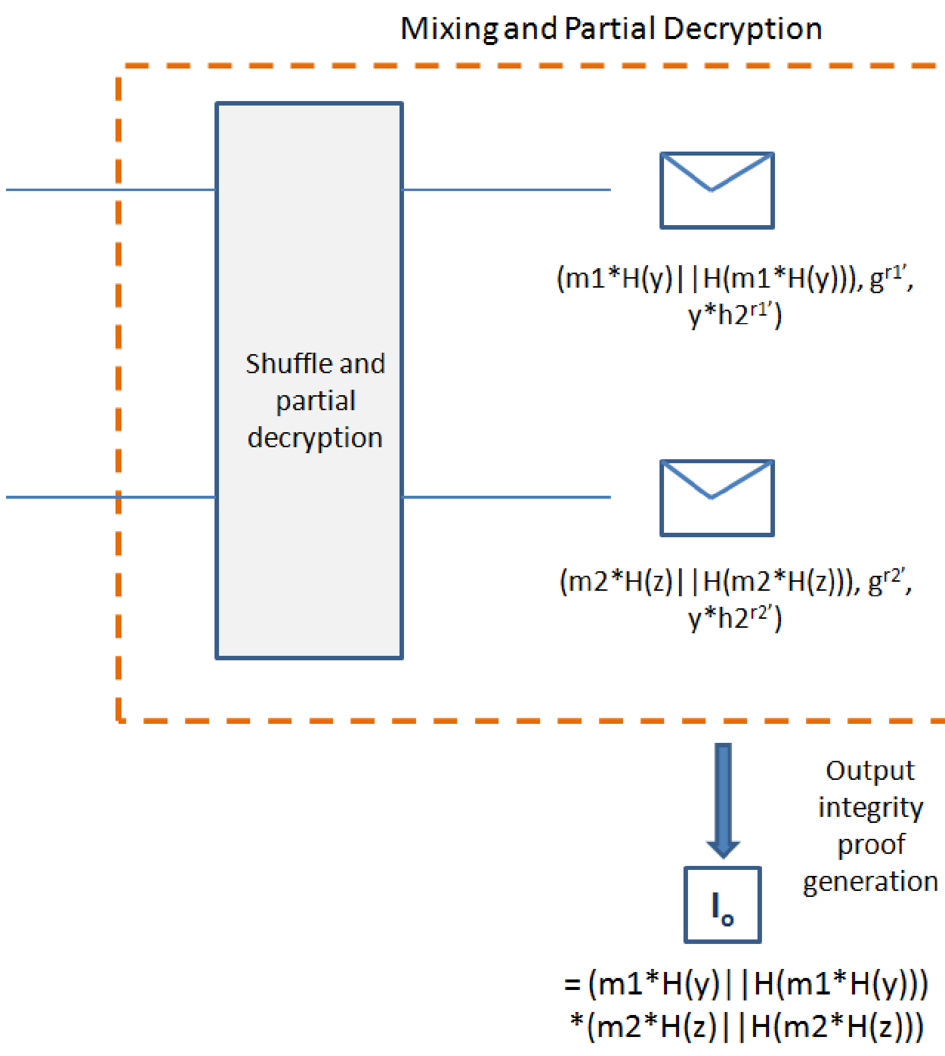

Mixing and Partial Decryption
Shuffle and partial decryption
(m1*H(y)||H(m1*H(y))), g^r1', y*h2^r1')
(m2*H(z)||H(m2*H(z))), g^r2', y*h2^r2')
Output integrity proof generation
I_o
= (m1*H(y)||H(m1*H(y))) *(m2*H(z)||H(m2*H(z)))

Fig. 4 - 15 Partial decryption and calculation of the *Output Integrity Proof*

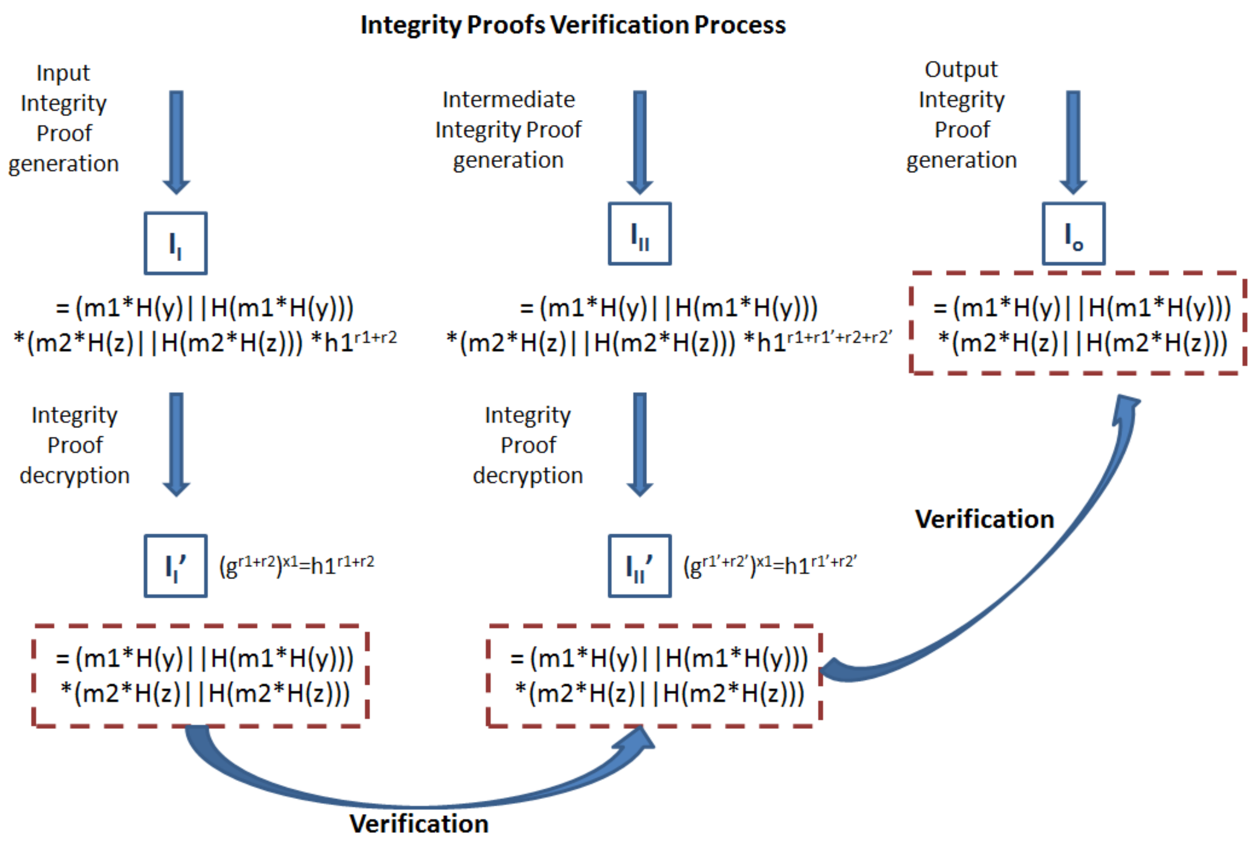

Integrity Proofs Verification Process
Input Integrity Proof generation
Intermediate Integrity Proof generation
Output Integrity Proof generation
I_I
I_{II}
I_o
$= (m1*H(y)||H(m1*H(y))) *(m2*H(z)||H(m2*H(z))) *h1^{r1+r2}$
$= (m1*H(y)||H(m1*H(y))) *(m2*H(z)||H(m2*H(z))) *h1^{r1+r1'+r2+r2'}$
$= (m1*H(y)||H(m1*H(y))) *(m2*H(z)||H(m2*H(z)))$
Integrity Proof decryption
Integrity Proof decryption
I_I'
$(g^{r1+r2})^{x1}=h1^{r1+r2}$
I_{II}'
$(g^{r1'+r2'})^{x1}=h1^{r1'+r2'}$
$= (m1*H(y)||H(m1*H(y))) *(m2*H(z)||H(m2*H(z)))$
$= (m1*H(y)||H(m1*H(y))) *(m2*H(z)||H(m2*H(z)))$
Verification
Verification

Fig. 4 - 16 *Integrity Proofs* verification step

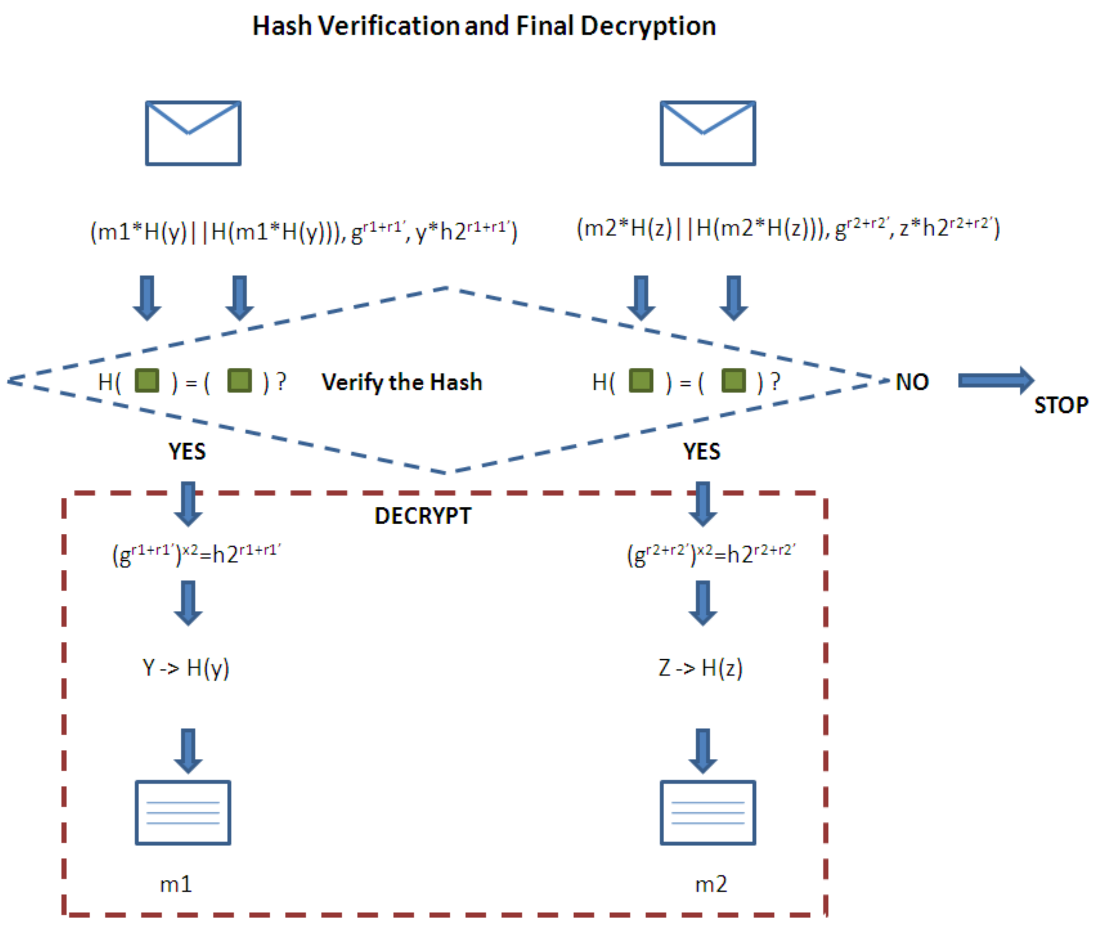

Hash Verification and Final Decryption
(m1*H(y)||H(m1*H(y))), g^{r1+r1'}, y*h2^{r1+r1'})
(m2*H(z)||H(m2*H(z))), g^{r2+r2'}, z*h2^{r2+r2'})
H( ) = ( ) ?
Verify the Hash
H( ) = ( ) ?
NO
STOP
YES
YES
DECRYPT
(g^{r1+r1'})^{x2}=h2^{r1+r1'}
(g^{r2+r2'})^{x2}=h2^{r2+r2'}
Y -> H(y)
Z -> H(z)
m1
m2

Fig. 4 - 17 Hash verification and final decryption

4.4.8.3. Performance of the Proposal 5

In order to check the performance of the Proposal 5 1024 bit-size keys and two input messages have been used.

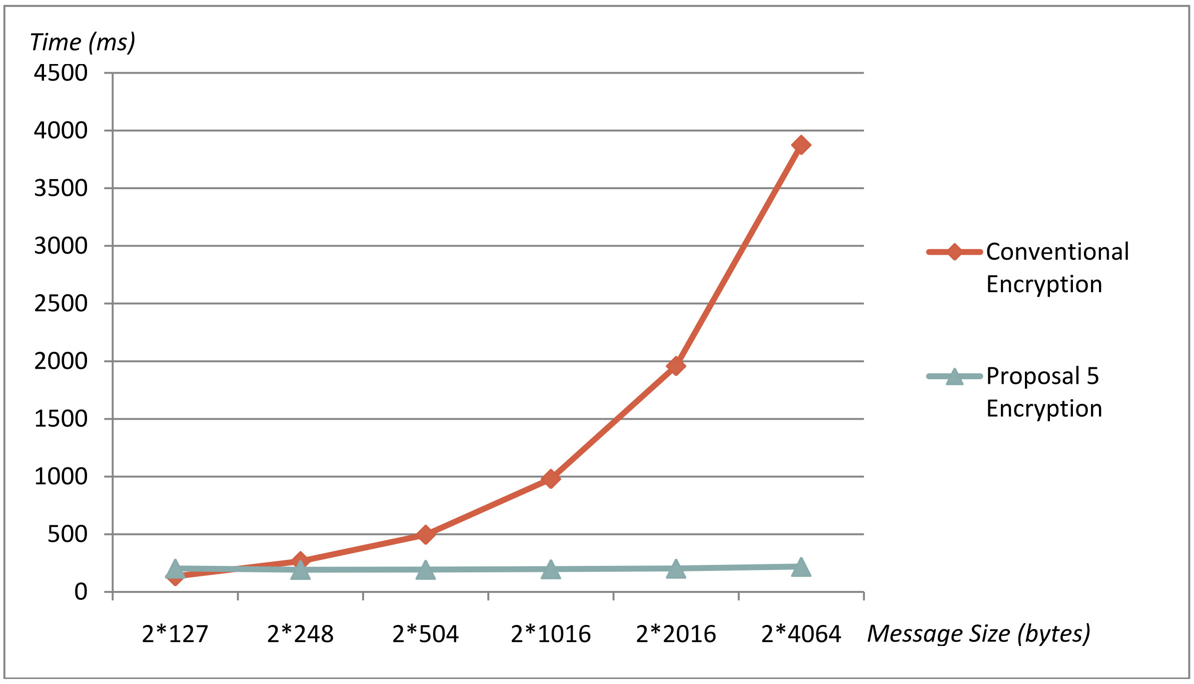


Fig. 4 - 18 Encryption time

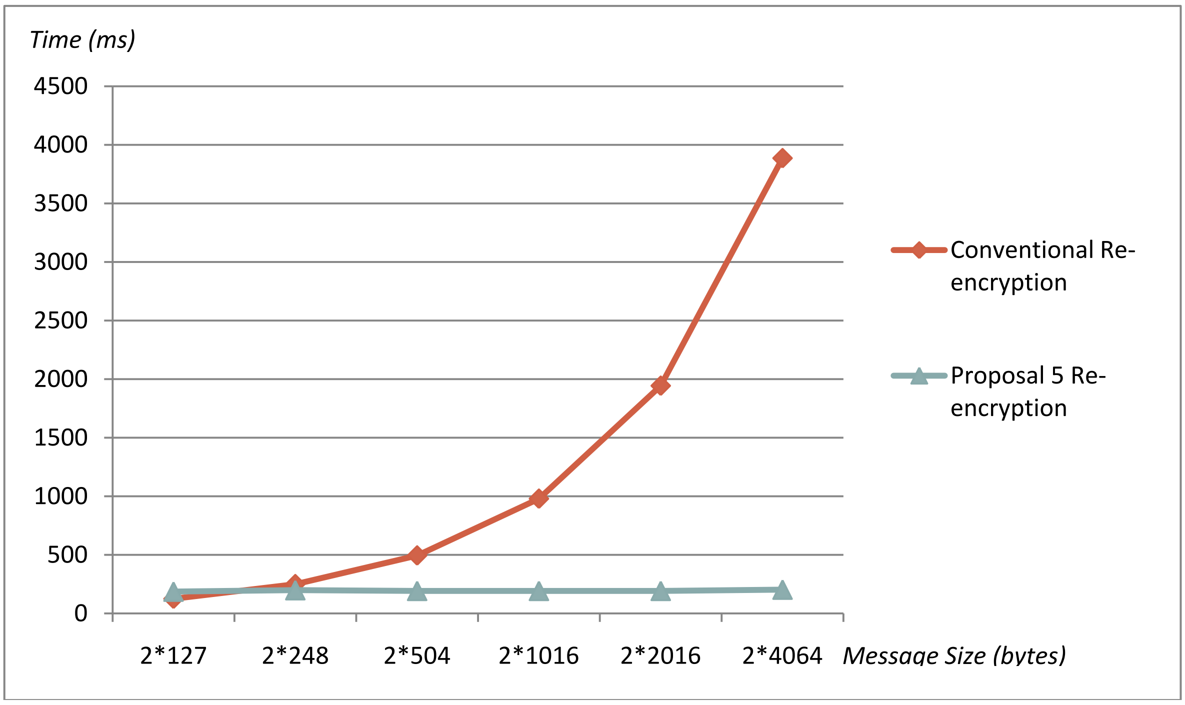


Fig. 4 - 19 Re-encryption time

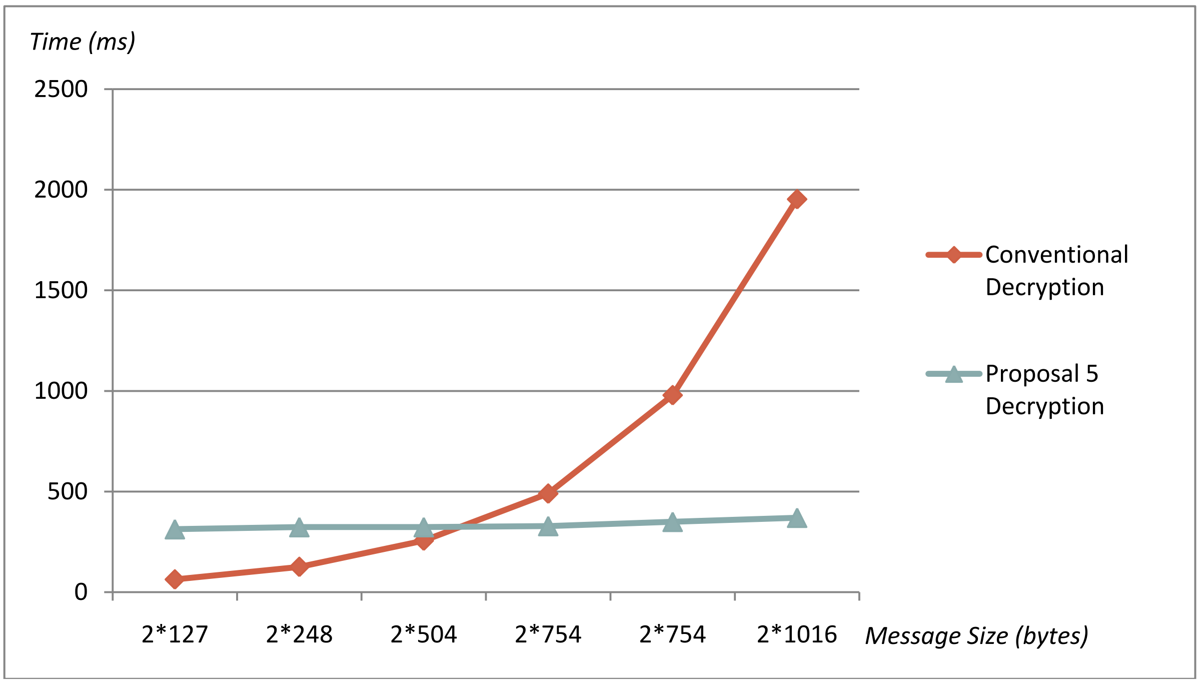


Fig. 4 - 20 Decryption time

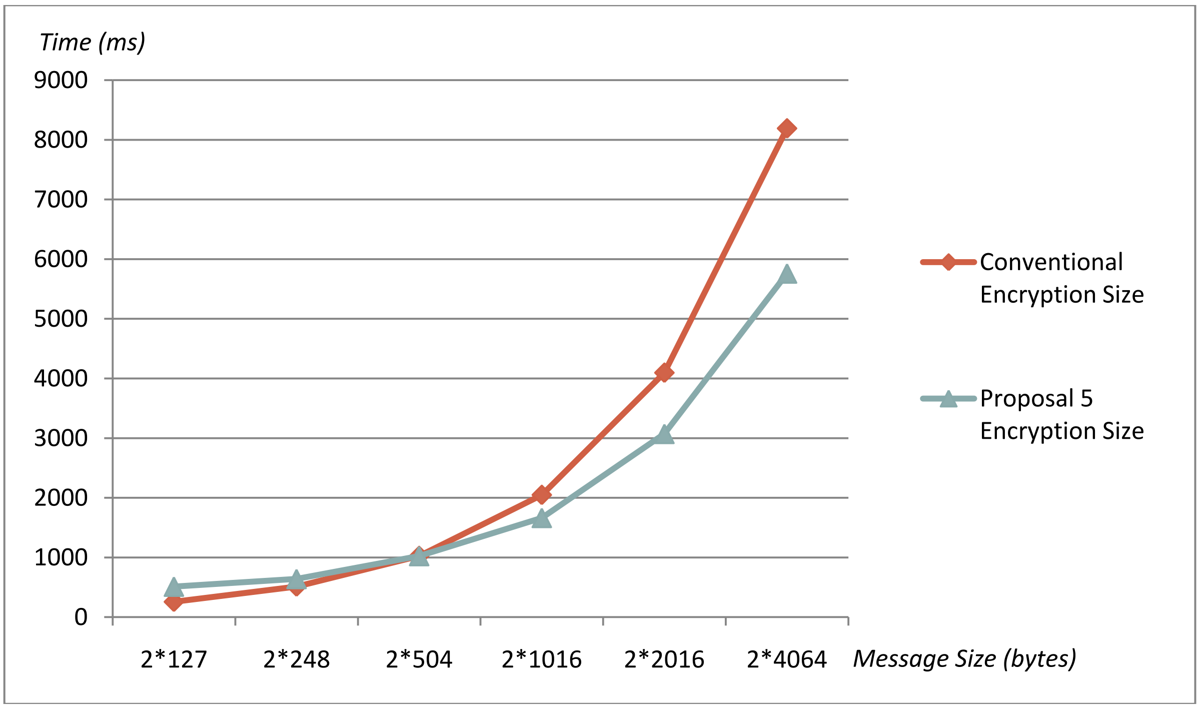


Fig. 4 - 21 Encryption size

These graphics show the time needed to encrypt, re-encrypt and decrypt two multi-block messages, comparing the performance of the conventional ElGamal cryptosystem and the one presented in the Proposal 5, including the calculation and verification of the *Integrity Proofs* and the individual Hash verification for each message block before the final decryption.

The performance of the Proposal 5 in the encryption/re-encryption/decryption processes is constant while the message size increases, in contraposition to the behaviour of the conventional ElGamal cryptosystem. This is due to the fact that the number of fixed operations (per message) in the Proposal 5 increases and the number of variable operations (per message block) decreases in comparison with the conventional cryptosystem, going on with the same philosophy that was the aim of the proposals 3 and 4.

Therefore, for small messages the performance of the conventional cryptosystem is better.

The decryption operation in the Proposal 5 is a little slower than in earlier proposals because the verification of the *Integrity Proofs* and the individual Hash checking for each message block are done in this process. Even so, the process is faster than when the conventional cryptosystem is used.

A resume of the fixed and variable operations for each cryptosystem is presented in Table 3:

	Fixed (for each message)	**Variable (for each message block)**
Conventional Encryption		1 random generation 2 modular exponentiations
Proposal 5 Encryption	2 random generation 3 modular exponentiations	2 hash calculations
Conventional Re-encryption		1 random generation 2 modular exponentiations
Proposal 5 Re-encryption	1 random generation 3 modular exponentiations	
Conventional Decryption		1 modular exponentiation 1 inversion
Proposal 5 Decryption	2 modular exponentiations 2 inversion	1 hash calculation 1 inversion

Table 3

Although the encryption size still increases faster with the message size using the conventional encryption than using the Proposal 5, the fact of adding a Hash resume next to the block message to be encrypted results in more block messages for the same message size, since the block size limitation is $[m_i+s_H] < q$ (where s_H is the size of the Hash resume) in the Proposal 5 instead of $m_i< q$ in the other proposals.

Due that more block message encryptions are needed in the Proposal 5 than in the conventional cryptosystem for the same message size, the encryption size for small messages is higher in the new proposal, and, although it is smaller than the encryption size in the conventional cryptosystem when the message size increases, it is bigger than in the other proposals.

In the figure 4.22 a comparison between the sizes of an encrypted message using the conventional cryptosystem and using the Proposal 5 is shown. The comparison is done encrypting a 2-block message (considering that the message blocks for the Proposal 5 are smaller than the ones in the conventional cryptosystem).

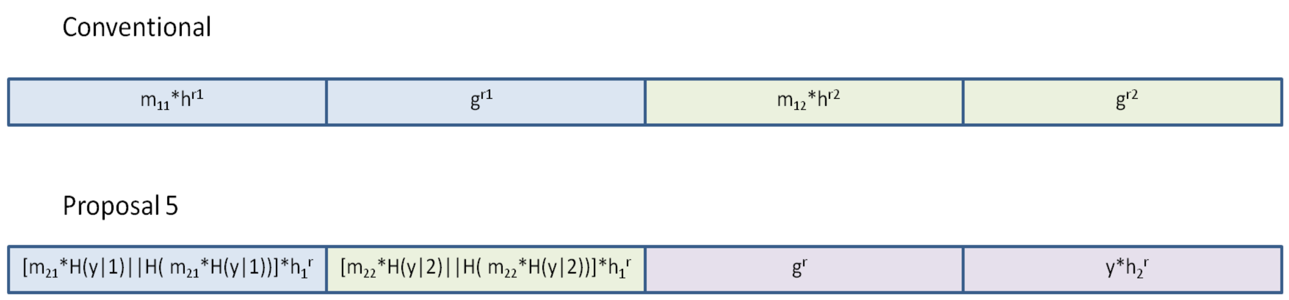

Conventional
$m_{11}*h^{r1}$
g^{r1}
$m_{12}*h^{r2}$
g^{r2}
Proposal 5
$[m_{21}*H(y|1)||H(m_{21}*H(y|1))]*h_1^r$
$[m_{22}*H(y|2)||H(m_{22}*H(y|2))]*h_1^r$
g^r
$y*h_2^r$

Fig. 4 - 22 Comparison of encryption sizes

4.4.8.4. Problems found in the Proposal 5

The problems found in the proposals 3 and 4 were solved in the Proposal 5, but another problem appears when this cryptosystem is used to encrypt messages that are processed in a mixnet in order to be de-correlated from their origin.

Although the relationship between the different parts of the cryptosystem *(a, b, d)* change which each re-encryption process in order to achieve non-traceable secret paths for the messages in the mix-nodes, the block encryptions of a same message have a constant relationship through the re-encryption processes:

Being each block message encrypted as

$$a_i = (m_i \cdot H(y,i)||H(m_i \cdot H(y,i))) \cdot {h_1}^r$$

The relationship between two encrypted block messages is

$$\frac{a_1}{a_2} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1))) \cdot {h_1}^r}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2))) \cdot {h_1}^r} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1)))}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2)))}$$

After a re-encryption process, each block message is encrypted as

$$a_i' = (m_i \cdot H(y,i)||H(m_i \cdot H(y,i))) \cdot {h_1}^{r+r'}$$

And then, the relationship between two encrypted block messages is

$$\frac{a_1'}{a_2'} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1))) \cdot {h_1}^{r+r'}}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2))) \cdot {h_1}^{r+r'}} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1)))}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2)))}$$

The relationship between two message blocks remains constant after the re-encryption. Therefore, the attacker could track the message through the different shuffling and re-encryption processes searching for the same relationship values at the input and the output of each mix-node as it is shown in figure 4.23.

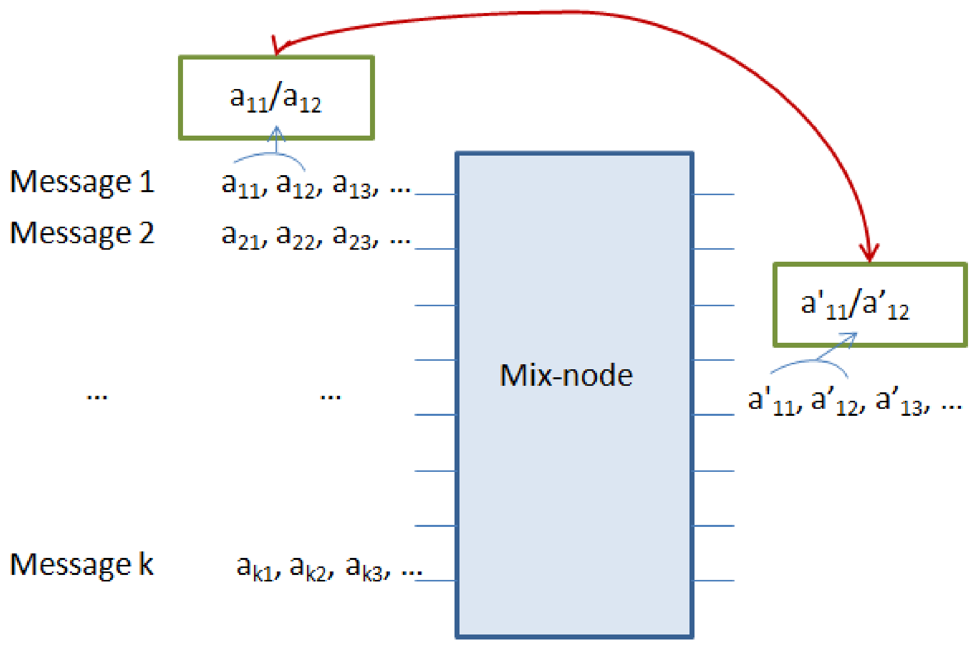


Fig. 4 - 23 Attack based on the re-encryption process

Therefore, the relationship between two block messages should not be constant. We have to change the system where all the blocks are encrypted using the same random factor ${h_1}^r$ ($a_i = (m_i \cdot H(y,i)||H(m_i \cdot H(y,i))) \cdot {h_1}^r$).

If we use a different random number *r* for each message block

$$a_i = (m_i \cdot H(y,i)||H(m_i \cdot H(y,i))) \cdot {h_1}^{r_i},$$

we lose the speediness that gives to us the fact of calculating just one exponentiation for all the message blocks, in front of what is needed in the conventional ElGamal cryptosystem. Thus, the efficiency goes down.

An alternative to this solution is presented in Proposal 6, where a unique exponentiation (${h_1}^r$) for all the message blocks to be encrypted is combined with a new exponentiation (${r_2}^{H(i)}$) for each one in the re-encryption process with a smaller exponent.

While the encryption exponent *r* has the same magnitude of the public encryption factor *p*, that must be at least a 1024-bit size number (better 2048-bit) [18], the “dispersion” exponent $H(i)$ is the result of a hash function, that usually has a smaller size than the module of a cryptosystem based on the DLP (256, 512 bits…), so a modular exponentiation for each block using this type of exponents makes the Proposal 6 still faster than the conventional encryption.

Thus, the cryptosystem security is preserved by multiplying the cleartexts by ${h_1}^r$ while the untraceability of the messages through the mixnet is ensured by the component ${r_2}^{H(i)}$ calculated for each message block.

The encryption of the message information jointly with its hash that was designed in Proposal 5 is maintained in this proposal; therefore, the verification of the encrypted message integrity block per block can be performed before it is finally decrypted.

4.4.9. Proposal 6

Key Generation

- Generate 3 secret keys x_1, x_2, x_3 as random numbers in *Zq*.

- The public keys are:

  $h_1=g^{x1}$

  $h_2=g^{x2}$

  $h_3=g^{x3}$

As in the Proposal 5, the secret keys can be given to different organizations in order to increase the system trust.

Encryption

- Each message *m* is split in blocks m_i so that $[m_i+s_H] < q$, where s_H is the size of a hash function that will be attached to the message block.

- Two random parameters (different for each message, equal for all the blocks of a message) $\{r_1, y\} < q$ are calculated.

- For each m_i:

  - Calculate $H(y,i)$, where *i* is the message block index.

  - Calculate $H\big(m_i \cdot H(y,i)\big)$.

  - Construct $a_i = [m_i \cdot H(y,i)||H\big(m_i \cdot H(y,i)\big)]$.

- The encrypted message is equal to:

$$(\{a_i, \dots, a_n\} \cdot {h_1}^{r_1}, g^{r_1}, {h_2}^{r_1} \cdot y, {h_3}^{r_1}) = (a, b, d, e).$$

Re-encryption

- Calculate two random parameters in *Zq*, r_1' and r_2'.

- Perform the re-encryption process as:

  - $b' = b \cdot g^{r_1'}$.

  - $d' = d \cdot {h_2}^{r_1'}$.

  - $e' = e \cdot {h_3}^{r_1'} \cdot r_2'$.

  - $a' = a \cdot {h_1}^{r_1'} \cdot \{{r_2}'^{H(i)}\}$,

    where the operation

$$a_i' = [m_i \cdot H(y,i)||H\big(m_i \cdot H(y,i)\big)] \cdot {h_1}^{r_1+r_1'} \cdot {r_2}'^{H(i)}$$

is performed for each block.

- The re-encryption result is:

$$\left(\left\{a_1 \cdot {r_2}'^{H(1)}, \dots, a_n \cdot {r_2}'^{H(n)}\right\} \cdot {h_1}^{r_1+r_1'}, g^{r_1+r_1'}, {h_2}^{r_1+r_1'} \cdot y, {h_3}^{r_1+r_1'} \cdot {r_2}'\right) = a', b', d', e'$$

where *n* is the number of message blocks.

As in the Proposal 5, a partial decryption process is done before the complete decryption in order to verify the hash attached to each message block.

Partial Decryption

- Calculate in the reception: ${h_3}^{r_1+r_1'} = (g^{r_1+r_1'})^{x_3}$ in order to obtain $r_2' = e' \cdot (g^{r_1+r_1'})^{-x_3}$.

- Calculate ${h_1}^{r_1+r_1'} = (g^{r_1+r_1'})^{x_1}$.

- Calculate ${r_2}'^{H(i)}$ for *i=1..n*.

- Then obtain

$$a_i = m_i \cdot H(y,i) || H(m_i \cdot H(y,i)) = \frac{a_i'}{{h_1}^{r1+r1'} + {r_2}'^{H(i)}}$$

for each message block.

Hash Verification

If we have the component

$$a_i = m_i \cdot H(y,i) || H(m_i \cdot H(y,i)),$$

for each encrypted message or message block:

- The verifier divides it in two pieces:

$$s = m_i \cdot H(y,i)$$

$$t = H(m_i \cdot H(y,i))$$

- Then checks that *H(s)=t*.

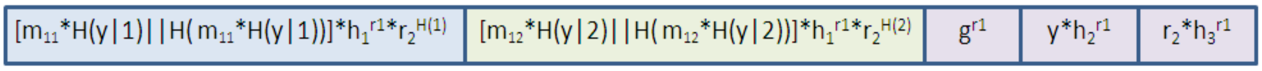
If the verification process succeeds, the process can continue. Otherwise it is stopped and an error is reported.

Decryption:

- Calculate ${h_2}^{r_1+r_1'} = (g^{r_1+r_1'})^{x_2}$ in order to obtain $y = d' \cdot (g^{r_1+r_1'})^{-x_2}$
- For each encrypted message block a_{subi} obtain m_i as

  $[m_i \cdot H(y,i)] \cdot H(y,i)^{-1}$.

An example of an encrypted message in the Proposal 6 is:


[m11*H(y|1)||H(m11*H(y|1))]*h1^r1*r2^H(1)
[m12*H(y|2)||H(m12*H(y|2))]*h1^r1*r2^H(2)
g^r1
y*h2^r1
r2*h3^r1
Fig. 4 - 24 Encrypted message using the Proposal 6

Critical operations needed:

- Encryption: two random number generations and four exponentiations at first.
  - $g^{r_1}, {h_1}^{r_1}, {h_2}^{r_1}, {h_3}^{r_1}, y$

  Two Hash functions are calculated for each message block.
  - $H(y,i), H\big(m_i \cdot H(y,i)\big)$

- Re-encryption: two random number generations and four exponentiations for each message.
  - $g^{r_1'}, {h_1}^{r_1'}, {h_2}^{r_1'}, {h_3}^{r_1'}, r_2'$

  One modular exponentiation with small exponent is needed to re-encrypt each message block.
  - ${r_2}'^{H(i)}$

- Decryption: three modular exponentiations are calculated for each message.
  - ${h_1}^{r_1+r_1'} = (g^{r_1+r_1'})^{x_1}$
  - ${h_2}^{r_1+r_1'} = (g^{r_1+r_1'})^{x_2}$
  - ${h_3}^{r_1+r_1'} = (g^{r_1+r_1'})^{x_3}$

  One Hash function and one modular exponentiation with small exponent are needed to decrypt each message block.
  - $H(y,i), {r_2}'^{H(i)}$

- Verification: a hash function is needed in order to verify the integrity of each message block.
  - $H\big(m_i \cdot H(y,i)\big)$

Traceability Prevention

Using the dispersion method proposed in the re-encryption process we prevent the message traceability through the re-encryption nodes.

In the Proposal 5 we had each block message encrypted as

$$a_i = (m_i \cdot H(y,i)||H(m_i \cdot H(y,i))) \cdot {h_1}^r$$

The relationship between two encrypted block messages was

$$\frac{a_1}{a_2} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1))) \cdot {h_1}^r}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2))) \cdot {h_1}^r} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1)))}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2)))}$$

Due that after a re-encryption process, each block message was encrypted as

$$a_i' = (m_i \cdot H(y,i)||H(m_i \cdot H(y,i))) \cdot {h_1}^{r+r'}$$

The relationship between two encrypted blocs was the same than before the re-encryption process:

$$\frac{a_1'}{a_2'} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1))) \cdot {h_1}^{r+r'}}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2))) \cdot {h_1}^{r+r'}} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1)))}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2)))}$$

In the Proposal 6 the relationship between two encrypted blocks before re-encryption is

$$\frac{a_1}{a_2} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1))) \cdot {h_1}^{r_1}}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2))) \cdot {h_1}^{r_1}} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1)))}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2)))}$$

And after the re-encryption the relationship is

$$\frac{a_1}{a_2} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1))) \cdot {h_1}^{r_1+r_1'} \cdot {r_2}'^{H(1)}}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2))) \cdot {h_1}^{r_1+r_1'} \cdot {r_2}_{'}^{H(2)}}$$
$$= \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1))) \cdot {r_2}'^{H(1)}}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2))) \cdot {r_2}'^{H(2)}}$$

If we re-encrypt again the message, the relationship between the two blocks is

$$\frac{a_1}{a_2} = \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1))) \cdot {h_1}^{r_1+r_1'+r_1''} \cdot ({r_2}''+{r_2}')^{H(1)}}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2))) \cdot {h_1}^{r_1+r_1'+r_1''} \cdot ({r_2}''+{r_2}')^{H(2)}}$$
$$= \frac{(m_1 \cdot H(y,1)||H(m_1 \cdot H(y,1))) \cdot ({r_2}''+{r_2}')^{H(1)}}{(m_2 \cdot H(y,2)||H(m_2 \cdot H(y,2))) \cdot ({r_2}''+{r_2}')^{H(2)}}$$

Thus, since the random numbers r_2 and ${r_2}'$ are not known by an external attacker, the relationship between the encrypted blocks changes at each re-encryption step and the messages can not be traced.

How could we use the Integrity Proofs?

As in the previous proposals, here we will explain how can we calculate and compare *Integrity Proofs* in order to verify a mixing process using this new proposal. Following the procedure followed in the Proposal 5, a scenario with multiple messages, each one with multiple blocks is presented:

<u>Input Integrity Proof Generation</u>

- From a set of input messages in a process (mixing), each one being encrypted as

  $(\{a_i, \dots, a_n\}, g^{r_1}, {h_2}^{r_1} \cdot y, {h_3}^{r_1})$,

  where $a_i = (m_i \cdot H(y,i) || H(m_i \cdot H(y,i))) \cdot {h_1}^{r}$

- An *Input Integrity Proof* is calculated as

$$I_I = \left(\prod_{j=1}^{k} \prod_{i=1}^{n} a_{ij}, \prod_{j=1}^{k} \prod_{i=1}^{n} g^{r_{1j}} \right)$$
$$= \left(\prod_{j=1}^{k} \prod_{i=1}^{n} (m_{ij} \cdot H(y_j,i) || H\left(m_{ij} \cdot H(y_j,i)\right)) \cdot {h_1}^{r_{1j}}, \prod_{j=1}^{k} \prod_{i=1}^{n} g^{r_{1j}} \right)$$

  Where *j=1…k*, being *k* the number of input messages and *i=1…n*, being *n* the number of blocks of each input message.

This proof is calculated based on all the blocks of the set of messages in the process (mixing) input.

<u>Intermediate Integrity Proof Generation</u> (calculated from the inputs or the outputs of an intermediate node)

- Being each encrypted message in the mixing process re-encrypted as

  $\left(\{a_1 \cdot {r_2}'^{H(1)}, \dots, a_n \cdot {r_2}'^{H(n)}\} \cdot {h_1}^{r_1'}, g^{r_1 + r_1'}, {h_2}^{r_1 + r_1'} \cdot y, {h_3}^{r_1 + r_1'} \cdot {r_2}'\right)$,

  where ${a_i}' = (m_i \cdot H(y,i) || H(m_i \cdot H(y,i))) \cdot {h_1}^{r_1 + r_1'} \cdot {r_2}'^{H(i)}$

- An *Intermediate Integrity Proof* is calculated as

$$I_{II} = \left(\prod_{j=1}^{k} \prod_{i=1}^{n} a_{ij}', \prod_{j=1}^{k} \prod_{i=1}^{n} g^{(r_1 + r_1')_j}, \prod_{j=1}^{k} g^{(r_1 + r_1')_j}, \prod_{j=1}^{k} {h_3}^{(r_1 + r_1')_j} \cdot {r_{2j}}' \right)$$
$$= \left(\prod_{j=1}^{k} \prod_{i=1}^{n} (m_{ij} \cdot H(y_j,i) || H\left(m_{ij} \cdot H(y_j,i)\right)) \cdot {h_1}^{(r_1 + r_1')_j} \right.$$
$$\left. \cdot {r_{2j}}'^{H(i)}, \prod_{j=1}^{k} \prod_{i=1}^{n} g^{(r_1 + r_1')_j}, \prod_{j=1}^{k} g^{(r_1 + r_1')_j}, \prod_{j=1}^{k} {h_3}^{(r_1 + r_1')_j} \cdot {r_{2j}}' \right)$$

  Where *j=1…k*, being *k* the number of input messages and *i=1…n*, being *n* the number of blocks of each input message.

This proof is calculated based on all the blocks of the set of messages in the process (mixing) input.

Output Integrity Proof Generation

- From the set of messages in the process (mixing) output, where each message block is partially decrypted as

$$m_i \cdot H(y,i)||H\big(m_i \cdot H(y,i)\big)$$

- The *Output Integrity Proof* can be calculated as

$$I_O = \left(\prod_{j=1}^{k}\prod_{i=1}^{n}[m_{ij} \cdot H(y_j,i))||H\left(m_{ij} \cdot H\big(y_j,i\big)\right)]\right)$$

  Where *j=1...k*, being *k* the number of output messages, *i=1…n*, being *n* the number of blocks of each output message.

This proof is calculated based on all the blocks of all the messages in the process (mixing) output.

Input Integrity Proof Decryption

In order to compare the *Input IntegrityProof* and the *Output Integrity Proof* in a process where the output messages are decrypted or partially decrypted, the *Input Integrity Proof* has to be decrypted:

- From the *Input Integrity Proof*

$$I_I = \left(\prod_{j=1}^{k}\prod_{i=1}^{n} a_{ij}, \prod_{j=1}^{k}\prod_{i=1}^{n} g^{r_j}\right)$$
$$= \left(\prod_{j=1}^{k}\prod_{i=1}^{n}(m_{ij} \cdot H(y_j,i)||H\left(m_{ij} \cdot H\big(y_j,i\big)\right)) \cdot {h_1}^{r_j}, \prod_{j=1}^{k}\prod_{i=1}^{n} g^{r_j}\right)$$

  the receiver calculates

$$\left(\prod_{j=1}^{k}\prod_{i=1}^{n}(g^{r_j})\right)^{x_1} = \left(\prod_{j=1}^{k}(g^{r_j \cdot n})\right)^{x_1} = \prod_{j=1}^{k}\prod_{i=1}^{n}({h_1}^{r_j}) = \prod_{j=1}^{k}({h_1}^{r_j \cdot n})$$

- Then obtains the *Input Integrity Proof* decrypted as

$$I_I' = \left(\prod_{j=1}^{k}\prod_{i=1}^{n}[m_{ij} \cdot H(y_j,i)||H\left(m_{ij} \cdot H\big(y_j,i\big)\right)]\right)$$
$$= I_I(first\ element) \cdot \left(\prod_{j=1}^{k}({h_1}^{r_j * n})\right)^{-1}$$

Intermediate Integrity Proof Decryption

If we want to compare the *Intermediate Integrity Proof* with the *Output Integrity Proof*, we have to decrypt this proof too:

- From the *Intermediate Integrity Proof*

$$I_{II} = \left(\prod_{j=1}^{k} \prod_{i=1}^{n} a_{ij}, \prod_{j=1}^{k} \prod_{i=1}^{n} g^{(r_1+r_1')_j}, \prod_{j=1}^{k} g^{(r_1+r_1')_j}, \prod_{j=1}^{k} h_3{}^{(r_1+r_1')_j} \cdot r_{2j}' \right)$$
$$= \left(\prod_{j=1}^{k} \prod_{i=1}^{n} (m_{ij} \cdot H(y_j, i) || H\left(m_{ij} \cdot H(y_j, i)\right)) \cdot h_1{}^{(r_1+r_1')_j} \right.$$
$$\left. \cdot r_{2j}{}'^{H(i)}, \prod_{j=1}^{k} \prod_{i=1}^{n} g^{(r_1+r_1')_j}, \prod_{j=1}^{k} g^{(r_1+r_1')_j}, \prod_{j=1}^{k} h_3{}^{(r_1+r_1')_j} \cdot r_{2j}' \right)$$

the verifier calculates

$$P = \prod_{j=1}^{k} \prod_{i=1}^{n} \left(h_1{}^{(r_1+r_1')_j}\right) = \prod_{j=1}^{k} \left(h_1{}^{(r_1+r_1')_j \cdot n}\right) = \left(\prod_{j=1}^{k} \prod_{i=1}^{n} \left(g^{(r_1+r_1')_j}\right) \right)^{x_1}$$
$$= \left(\prod_{j=1}^{k} \left(g^{(r_1+r_1')_j \cdot n}\right) \right)^{x_1}$$

$$Q = \prod_{j=1}^{k} h_3{}^{(r_1+r_1')_j} = \left(\prod_{j=1}^{k} g^{(r_1+r_1')_j} \right)^{x_3}$$

$$R = \prod_{j=1}^{k} r_{2j}' = \left(\prod_{j=1}^{k} h_3{}^{(r_1+r_1')_j} \cdot r_{2j}' \right) \cdot (Q)^{-1}$$

$$S = (R)^{\sum_{i=1}^{n} H(i)}$$

- Then obtains the *Intermediate Integrity Proof* decrypted as

$$I_{II}' = \left(\prod_{j=1}^{k} \prod_{i=1}^{n} [m_{ij} \cdot H(y_j, i) || H\left(m_{ij} \cdot H(y_j, i)\right)] \right)$$
$$= I_{II}(first\ element) \cdot (P)^{-1} \cdot (S)^{-1}$$

Integrity Proofs Verification

The Output Integrity Proof I_O and the Decrypted Input and Intermediate Integrity Proofs I'_I and I'_{II} are compared. In case the messages were manipulated during the process (mixing), these three proofs would not match. If there are various mix-nodes and we calculate an *Integrity Proof* in the input and the output of each one, we can know which node or nodes have cheated comparing the decrypted proofs.

Procedure

Basically, the procedure to use this cryptosystem properly in a re-encryption mixnet is the same than in the Proposal 5, changing some details about the implementation:

1. Message Encryption
2. Input Integrity Proof Generation
3. Message Re-encryption
4. Intermediate Integrity Proof Generation
5. Partial Decryption
6. Output Integrity Proof Generation
7. Input and Intermediate Integrity Proof Decryption
8. Integrity Proofs Verification
9. Hash Verification
10. Final Decryption

In the figures 4.25, 4.26, 4.27, 4.28 an example of the mixing procedure for two messages with two blocks each one is described.

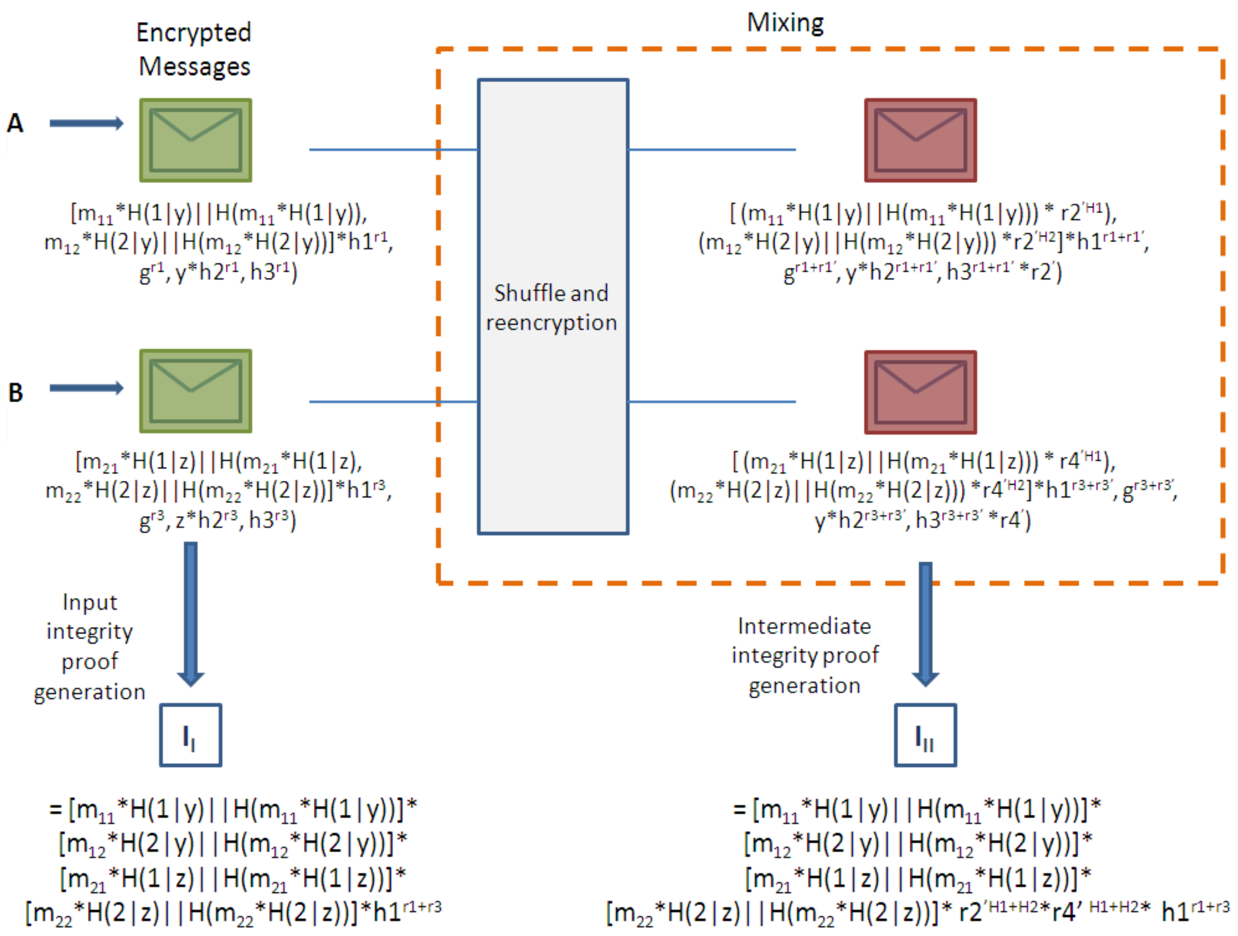

Encrypted Messages
Mixing
A
B
$[m_{11}*H(1|y)||H(m_{11}*H(1|y)), m_{12}*H(2|y)||H(m_{12}*H(2|y))]*h1^{r1}, g^{r1}, y*h2^{r1}, h3^{r1})$
$[m_{21}*H(1|z)||H(m_{21}*H(1|z), m_{22}*H(2|z)||H(m_{22}*H(2|z))]*h1^{r3}, g^{r3}, z*h2^{r3}, h3^{r3})$
Shuffle and reencryption
$[(m_{11}*H(1|y)||H(m_{11}*H(1|y))) * r2'^{H1}), (m_{12}*H(2|y)||H(m_{12}*H(2|y))) *r2'^{H2}]*h1^{r1+r1'}, g^{r1+r1'}, y*h2^{r1+r1'}, h3^{r1+r1'} *r2')$
$[(m_{21}*H(1|z)||H(m_{21}*H(1|z))) * r4'^{H1}), (m_{22}*H(2|z)||H(m_{22}*H(2|z))) *r4'^{H2}]*h1^{r3+r3'}, g^{r3+r3'}, y*h2^{r3+r3'}, h3^{r3+r3'} *r4')$
Input integrity proof generation
Intermediate integrity proof generation
I_I
I_{II}
$= [m_{11}*H(1|y)||H(m_{11}*H(1|y))]* [m_{12}*H(2|y)||H(m_{12}*H(2|y))]* [m_{21}*H(1|z)||H(m_{21}*H(1|z))]* [m_{22}*H(2|z)||H(m_{22}*H(2|z))]*h1^{r1+r3}$
$= [m_{11}*H(1|y)||H(m_{11}*H(1|y))]* [m_{12}*H(2|y)||H(m_{12}*H(2|y))]* [m_{21}*H(1|z)||H(m_{21}*H(1|z))]* [m_{22}*H(2|z)||H(m_{22}*H(2|z))]* r2'^{H1+H2}*r4'^{H1+H2}* h1^{r1+r3}$

Fig. 4 - 25 Mixing process: shuffling and re-encryption. Calculation of the *Input* and *Intermediate Integrity Proofs*

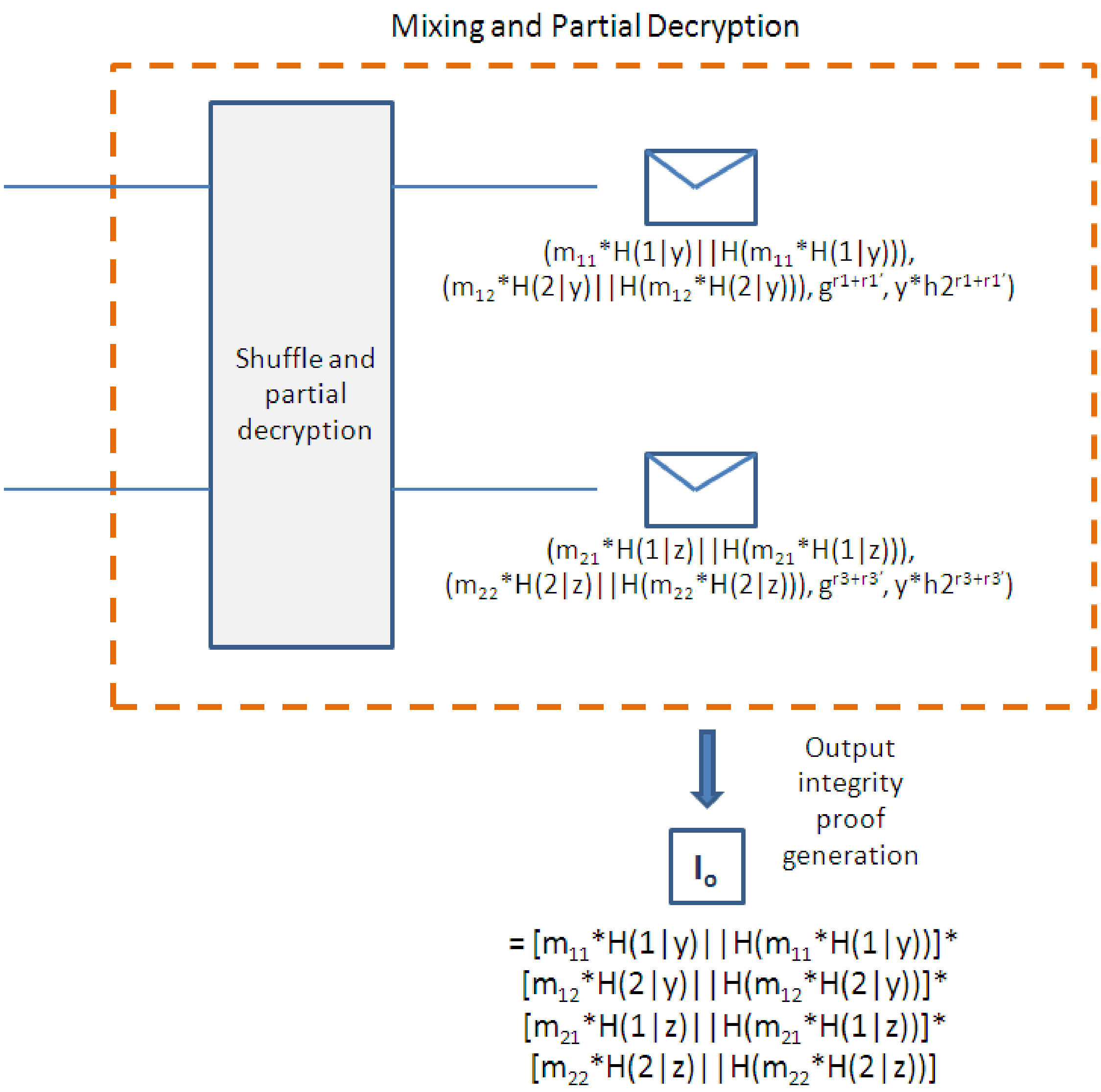

Mixing and Partial Decryption
Shuffle and partial decryption
$(m_{11}*H(1|y)||H(m_{11}*H(1|y)))$,
$(m_{12}*H(2|y)||H(m_{12}*H(2|y))), g^{r1+r1'}, y*h2^{r1+r1'})$
$(m_{21}*H(1|z)||H(m_{21}*H(1|z)))$,
$(m_{22}*H(2|z)||H(m_{22}*H(2|z))), g^{r3+r3'}, y*h2^{r3+r3'})$
Output integrity proof generation
I_o
$= [m_{11}*H(1|y)||H(m_{11}*H(1|y))]*$
$[m_{12}*H(2|y)||H(m_{12}*H(2|y))]*$
$[m_{21}*H(1|z)||H(m_{21}*H(1|z))]*$
$[m_{22}*H(2|z)||H(m_{22}*H(2|z))]$

Fig. 4 - 26 Partial decryption and *Output Integrity Proof* calculation

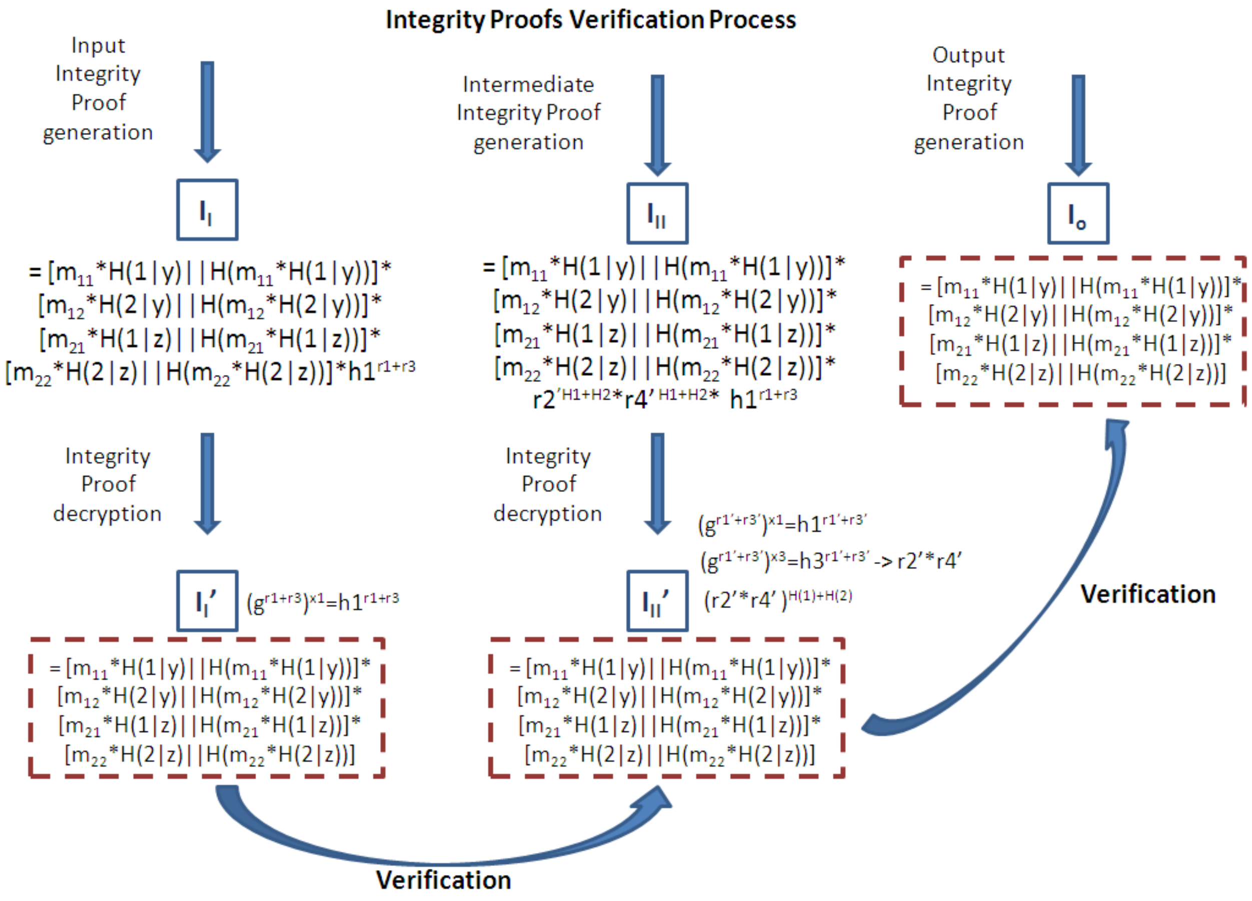

Integrity Proofs Verification Process
Input Integrity Proof generation
I_I
$= [m_{11}*H(1|y)||H(m_{11}*H(1|y))]* [m_{12}*H(2|y)||H(m_{12}*H(2|y))]* [m_{21}*H(1|z)||H(m_{21}*H(1|z))]* [m_{22}*H(2|z)||H(m_{22}*H(2|z))]*h1^{r1+r3}$
Integrity Proof decryption
I_I'
$(g^{r1+r3})^{x1}=h1^{r1+r3}$
$= [m_{11}*H(1|y)||H(m_{11}*H(1|y))]* [m_{12}*H(2|y)||H(m_{12}*H(2|y))]* [m_{21}*H(1|z)||H(m_{21}*H(1|z))]* [m_{22}*H(2|z)||H(m_{22}*H(2|z))]$
Verification
Intermediate Integrity Proof generation
I_{II}
$= [m_{11}*H(1|y)||H(m_{11}*H(1|y))]* [m_{12}*H(2|y)||H(m_{12}*H(2|y))]* [m_{21}*H(1|z)||H(m_{21}*H(1|z))]* [m_{22}*H(2|z)||H(m_{22}*H(2|z))]* r2'^{H1+H2}*r4'^{H1+H2}* h1^{r1+r3}$
Integrity Proof decryption
$(g^{r1'+r3'})^{x1}=h1^{r1'+r3'}$
$(g^{r1'+r3'})^{x3}=h3^{r1'+r3'} \rightarrow r2'*r4'$
$(r2'*r4')^{H(1)+H(2)}$
I_{II}'
$= [m_{11}*H(1|y)||H(m_{11}*H(1|y))]* [m_{12}*H(2|y)||H(m_{12}*H(2|y))]* [m_{21}*H(1|z)||H(m_{21}*H(1|z))]* [m_{22}*H(2|z)||H(m_{22}*H(2|z))]$
Output Integrity Proof generation
I_O
$= [m_{11}*H(1|y)||H(m_{11}*H(1|y))]* [m_{12}*H(2|y)||H(m_{12}*H(2|y))]* [m_{21}*H(1|z)||H(m_{21}*H(1|z))]* [m_{22}*H(2|z)||H(m_{22}*H(2|z))]$
Verification

Fig. 4 - 27 *Integrity Proofs* verification

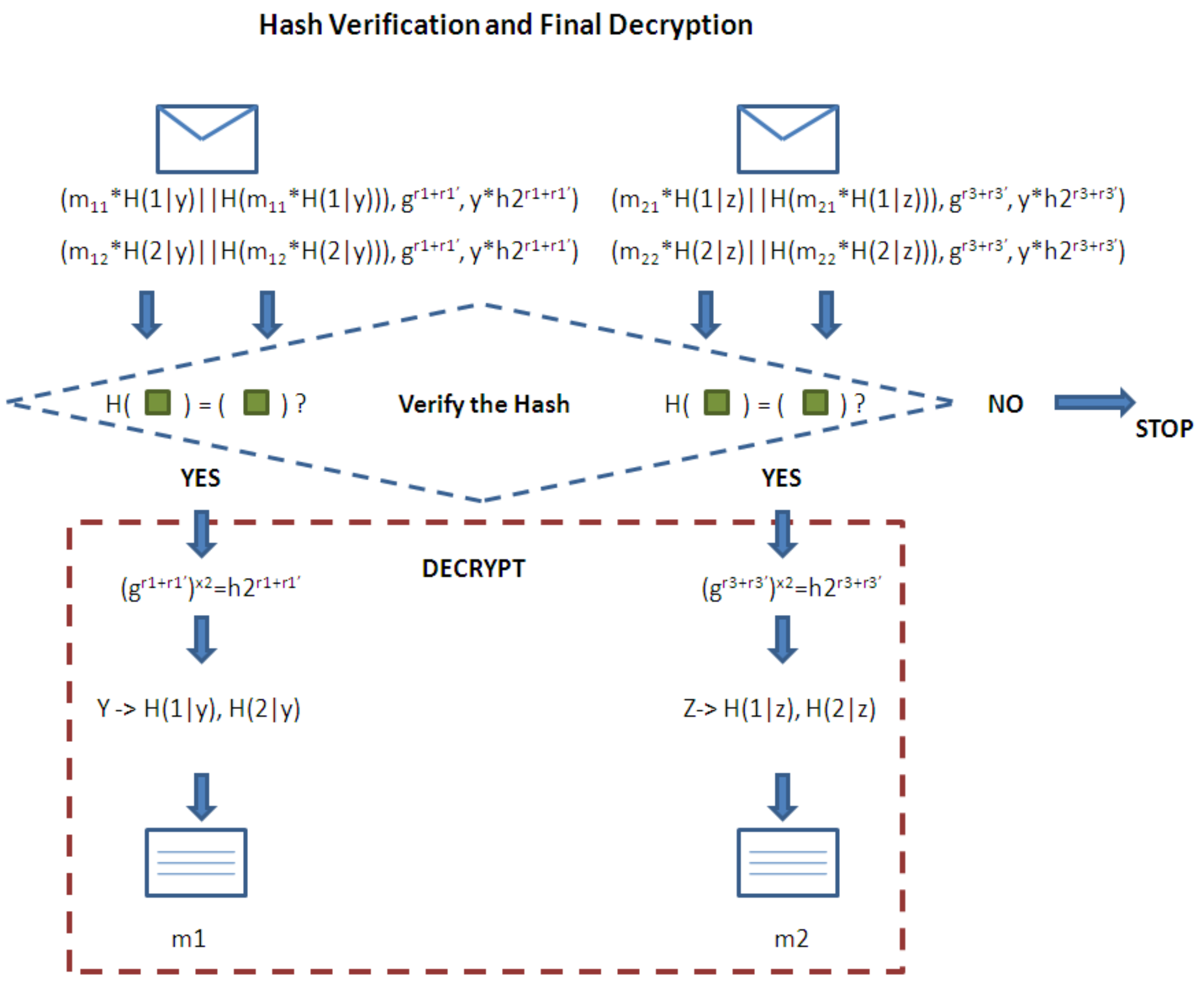

Hash Verification and Final Decryption
$(m_{11}*H(1|y)||H(m_{11}*H(1|y))), g^{r1+r1'}, y*h2^{r1+r1'})$
$(m_{12}*H(2|y)||H(m_{12}*H(2|y))), g^{r1+r1'}, y*h2^{r1+r1'})$
$(m_{21}*H(1|z)||H(m_{21}*H(1|z))), g^{r3+r3'}, y*h2^{r3+r3'})$
$(m_{22}*H(2|z)||H(m_{22}*H(2|z))), g^{r3+r3'}, y*h2^{r3+r3'})$
H(🟩) = (🟩) ?
Verify the Hash
H(🟩) = (🟩) ?
NO
STOP
YES
YES
DECRYPT
$(g^{r1+r1'})^{x2}=h2^{r1+r1'}$
$(g^{r3+r3'})^{x2}=h2^{r3+r3'}$
Y -> H(1|y), H(2|y)
Z-> H(1|z), H(2|z)
m1
m2

Fig. 4 - 28 Hash verification and final decryption

4.4.9.1. Performance of the Proposal 6

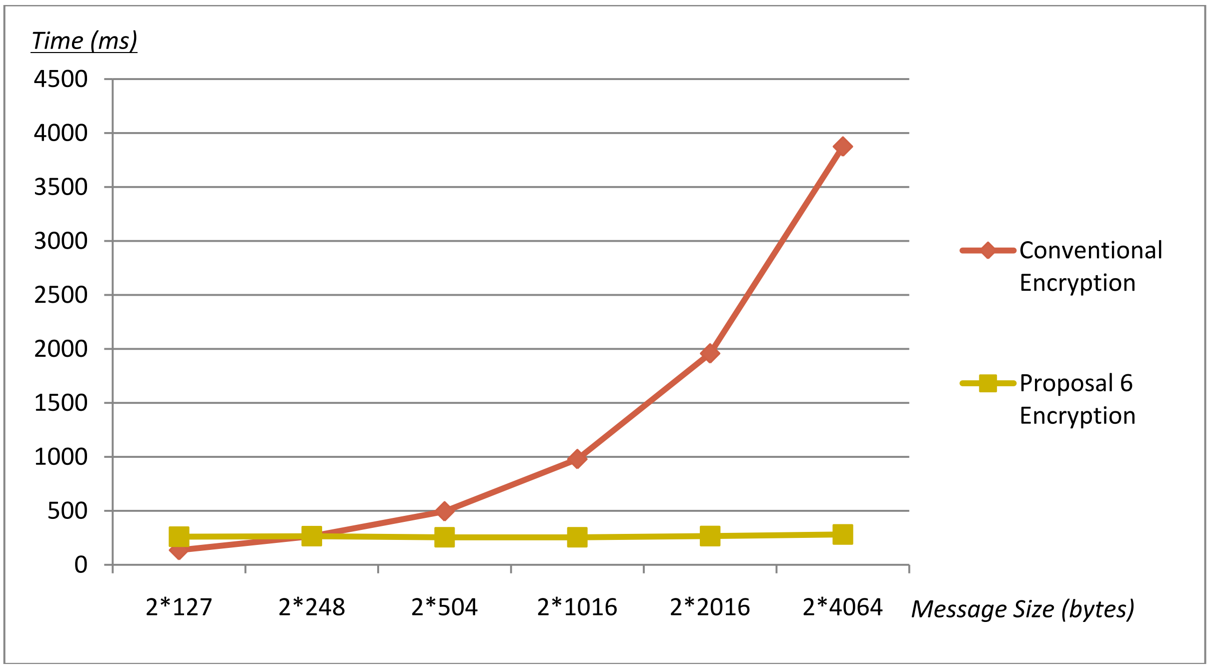


Fig. 4 - 29 Encryption time

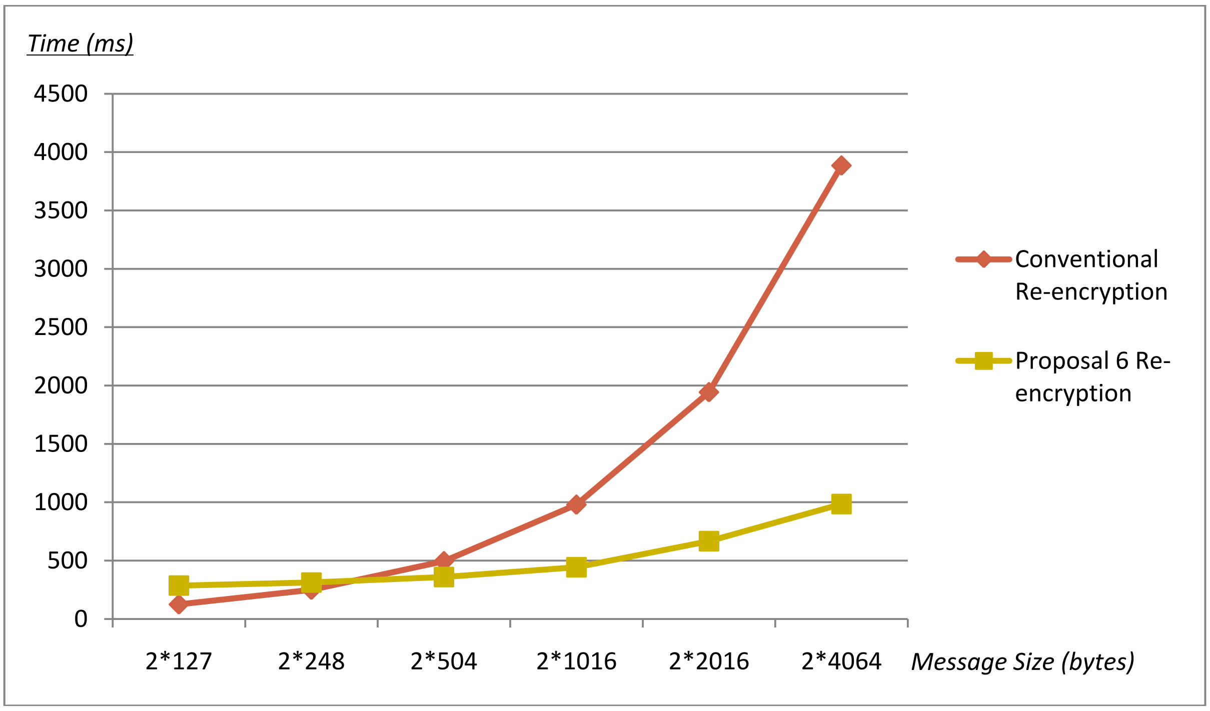


Fig. 4 - 30 Re-encryption time

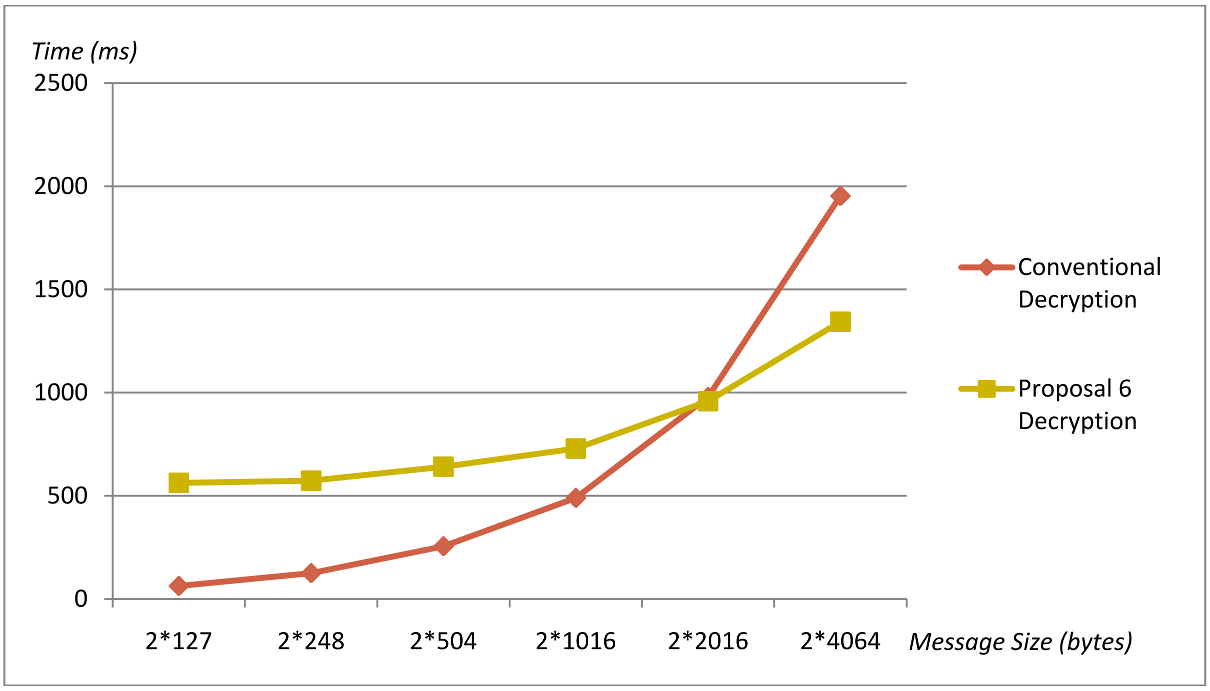


Fig. 4 - 31 Decryption time

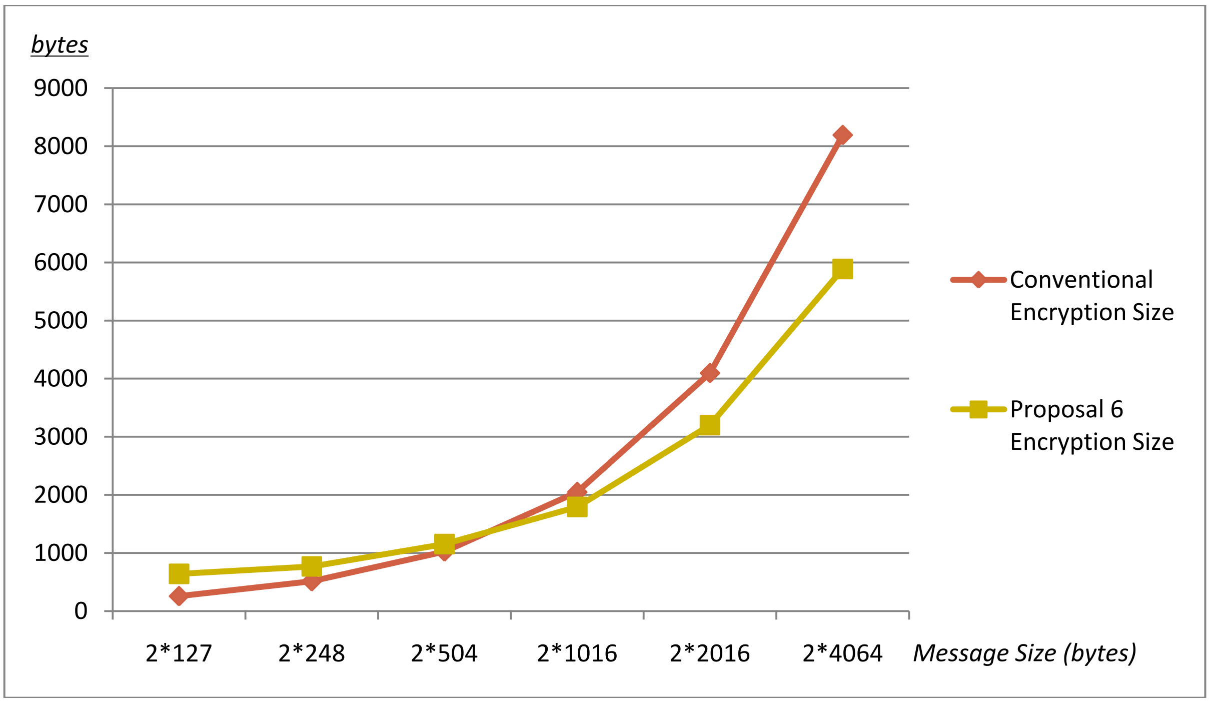


Fig. 4 - 32 Encryption size

The graphics show the time resources needed to encrypt, re-encrypt and decrypt two multi-block messages in both conventional and Proposal 6 cryptosystems. Like in the Proposal 5, the simulation in these graphics includes the time of calculation and verification of the *Integrity Proofs* and the individual Hash verification of each message block.

Unlike in the previous proposals, the re-encryption and decryption times in this proposal are not constant while the message size increases. This is due to the fact that, in order to prevent message tracing through the mix-nodes, a small modular exponentiation is performed for each encrypted block instead of performing just an exponetiation for each message as in the previous proposals. Since the exponentiation to be performed in the re-encryption and decryption processes uses a small exponent, this cryptosystem is still faster than the conventional one. Finally, we can say that we have left the heavy computations to the mixing and decryption processes, since the process for encryption (which has to be done by the voter) is still constant in time while the message size increases. We think this is a good approach, since we can not suppose that the user (the one that encrypts the message at first) will have a device fast enough to do the encryption process in a little time, but the mixing and decryption servers can be designed to support these heavy computations.

Like in the other cases, the performance of the conventional cryptosystem for small messages is better.

A resume of the fixed and variable operations for each cryptosystem is presented in Table 4:

	Fixed (for each message)	**Variable (for each message block)**
Conventional Encryption		1 random generation 2 modular exponentiations
Proposal 6 Encryption	2 random generation 4 modular exponentiations	2 hash calculations
Conventional Re-encryption		1 random generation 2 modular exponentiations
Proposal 6 Re-encryption	2 random generation 4 modular exponentiations	1 modular exponentiation (small exponent)
Conventional Decryption		1 modular exponentiation 1 inversion
Proposal 6 Decryption	3 modular exponentiations 3 inversions	1 hash calculation 2 inversions 1 modular exponentiation (small exponent)

Table 4

Since we have added another constant factor in the encryption (h_3) in the Proposal 6, the ciphertext size is larger than in the Proposal 5, but it is still smaller than in the conventional encryption for large messages since most of the parameters are fixed for whichever is the size of the message to be encrypted.

Like in the Proposal 5, the fact of adding a Hash resume next to the block message to be encrypted results in more block divisions for the same message size than in the conventional cryptosystem, since the block size limitation is $[m_i+s_H] < q$ (where s_H is the size of the Hash resume) instead of $m_i < q$. This drawback makes the cryptosystem to divide each message to be encrypted into more and smaller blocks, increasing the computation time.

4.4.10. Conclusions about the proposed cryptosystems

Finally we can evaluate the features achieved with each proposal in order to see the process evolution in Table 5:

	Proposal 3	Proposal 4	Proposal 5	Proposal 6
PI Calculation	YES	YES	YES	YES
Homomorphism	YES	YES	YES	YES
Re-encryption	NO	YES	YES	YES
Message Traceability Prevention	NO	NO	NO	YES
Three-Hundred Attack Detection	NO	NO	YES	YES

Table 5

Neither the Proposal 1 nor 2 had the homomorphic properties we needed in order to calculate the *Integrity Proofs*. This issue was solved in the Proposal 3, so we could expose how the *Integrity Proofs* could be calculated in each proposal.

The Proposal 3 did not be suitable to apply re-encryption processes over the encrypted messages. Therefore, this topic was solved in the Proposal 4.

The 300 Attack detection was solved in the Proposal 5, and finally in the Proposal 6 we solved the problem of message traceability through the re-encryptions performed at each mix-node.

Finally, a comparison of the performance of the presented proposals is shown in the next graphics:

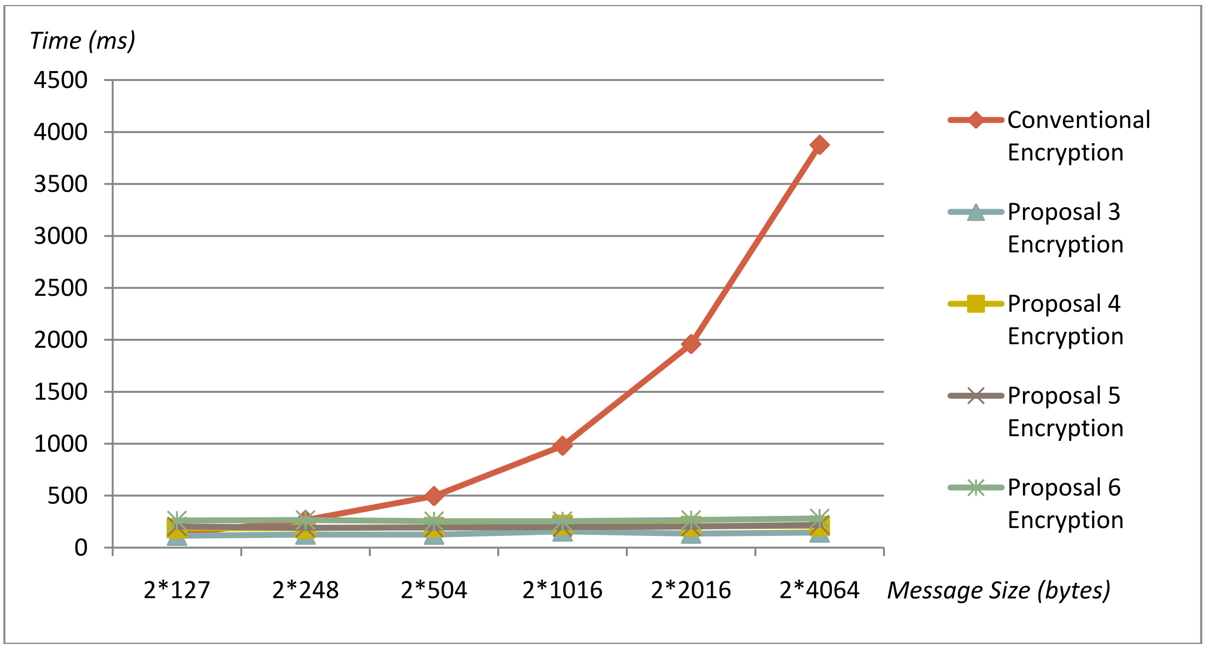


Fig. 4 - 33 Encryption time

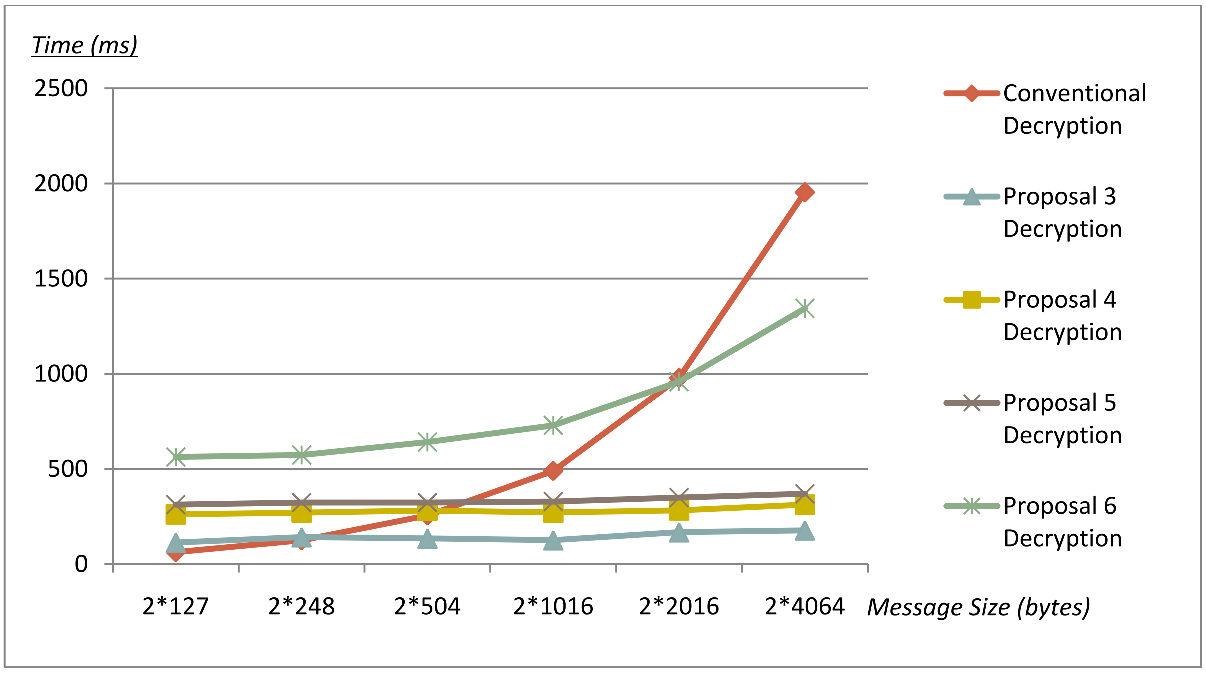


Fig. 4 - 34 Decryption time

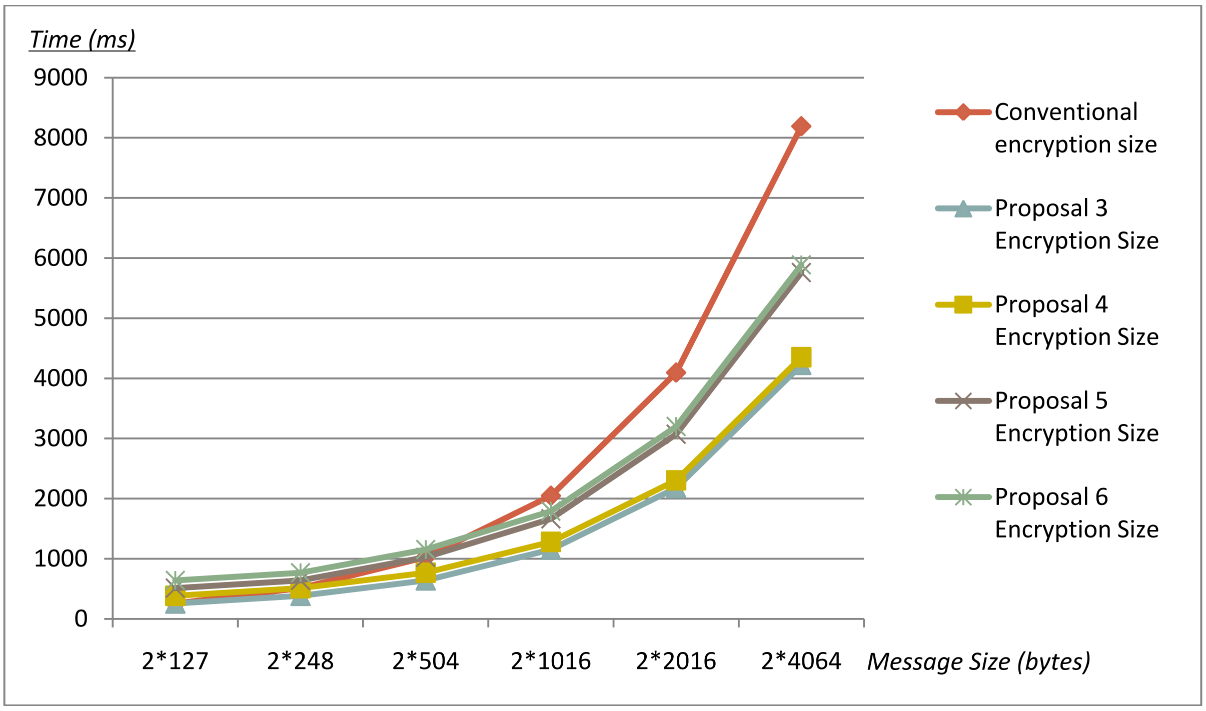


Fig. 4 - 35 Encryption size

The time needed in the encryption process is more or less the same for all the presented proposals, but in decryption time and encryption size the Proposal 6 is the worst of them, due to that we had to build a more complex scheme every time that a problem was detected in a proposal. Even so, the Proposal 6 is faster and requires less storage resources than the conventional cryptosystem.

4.5. Elliptic Curve Cryptography: EC ElGamal Cryptosystem

4.5.1. Elliptic Curve Cryptography

The mathematical theory of elliptic curves [28] [29][30][31] provides a class of finite groups that have proven quite suitable for cryptographic use. Koblitz and Miller proposed in 1985 the use of the group of points of an elliptic curve in public-key cryptography.

The security in the Elliptic Curve Cryptography (ECC) depends on the difficulty of the Elliptic Curve Discrete Logarithm Problem:

- Let *P* and *Q* be two points on an elliptic curve such that *k·P=Q*, where *k* is a scalar.
- Given *P* and *Q*, it is computationally infeasible to obtain *k*, if *k* is sufficiently large.

k is called to be the discrete logarithm of *Q* to the base *P*. The name of ECDLP is due to its equivalence with the one found in cryptosystems as ElGamal, due that the additive operations in EC are similar to the exponentiations in ElGamal.

If the chosen finite field is large, the discrete logarithm problem on elliptic curve abelian groups is believed to be more difficult than the corresponding problem in the underlying finite field's multiplicative group. Therefore, one main advantage of ECC is that the key size usually needed in cryptographic applications is smaller than the one needed using conventional finitie fields. For example, a 160-bit key in ECC is considered to be as secured as 1024-bit key in RSA.

4.5.1.1. Mathematical background

The mathematic operations in elliptic curves are defined over an elliptic curve characterised by *a* and *b*:

$$y^2 = x^3 + a \cdot x + b,$$

where $4 \cdot a^3 + 27 \cdot b^2 \neq 0$.

Each value of *a* and *b* generates a different elliptic curve. All points which satisfy the above equation plus a point at infinity lie on the elliptic curve. This set of points on such a curve compose an abelian group, with the point at infinity as identity element. If the coordinates *x* and *y* are chosen from a finite field, the solutions compose a finite abelian group.

The elliptic curve arithmetic is described in [28], [31] and in [30].

Use of finite fields

The elliptic curve operations over the real numbers are slow and inaccurate due to round-off error. Cryptographic operations need to be faster and accurate. To make operations on elliptic curve accurate and more efficient, the curve cryptography is defined over two finite fields:

- Prime field F_p
- Binary field F_2^m

The field is chosen with finitely large number of points suited for cryptographic operations.

EC on Prime field F_p

In finite fields F_p the EC definition and operations are defined under the *p* module, having then the basic equations as:

$$y^2 \: mod \: p = x^3 + a \cdot x + b \: mod \: p$$

$$4 \cdot a^3 + 27 \cdot b^2 \: mod \: p \neq 0,$$

where all the integers ϵ *{0, p-1}.*

All the operations such as addition, substraction, division, multiplication involve integers between *0* and *p-1*. The primer number *p* is chosen such that there is finitely large number of points on the elliptic curve to make the cryptosystem secure. SECG [34] specifies curves with *p* ranging between 112-425 bits.

EC on Binary field F_2^m

In binary field F_2^m , the elliptic curve equations are

$$y^2 + x \cdot y = x^3 + a \cdot x^2 + b,$$

where $b \neq 0$.

The elements of the finite field are integers of length at most *m* bits. These numbers can be considered as a binary polynomial of degree *m-1*. All the operations are done with at most *m-1* grade polynomials.

The *m* is chosen such that there is finitely large number of points on the elliptic curve to make the cryptosystem secure. SECG [34] specifies curves with *m* ranging between 113-571 bits.

In F_p the elements are integers $1 \leq x \leq p$, which are combined using modular arithmetic. The case of F_2^m is slightly more complicated: one obtains different representations of the field elements as bitstrings for each choice of irreducible binary polynomial *f(x)* of degree *m*.

In our implementation we are going to use elliptic curves over finite fields F_p defined for the *p* module.

4.5.1.2. Domain parameters

To use ECC all parties must agree on all the elements defining the elliptic curve, that is the domain parameters of the scheme:

- The field is defined by *p* in the prime case and the pair of *m* and *f* in the binary case.

- The elliptic curve is defined by the constants *a* and *b* used in its defining equation.

- Finally, the cyclic subgroup is defined by its generator or base point *G*. For cryptographic application, the order of *G*, that is the smallest non-negative number *n* such that $n \cdot G = 0$, must be prime.

  Since *n* is the size of the subgroup of $E(F_p)$ it follows from the Lagrange's theorem that the number $h \cdot \#E(F_p)$/n is integer, where $\#E(F_p)$ is the number of points of the elliptic curve.
  In cryptographic applications this number *h*, called cofactor, must be small ($h \leq 4$) and, preferably, *h=1*. If not, the curve would be vulnerable to the Pohlig-Hellman attack.

  The same can be applied for $E(F_2^m)$

In the prime case the domain parameters are *(p, a, b, G, n, h)* and in the binary case they are *(m, f, a, b, G, n, h)*.

Finally, not all the elliptic curves are useful in cryptographic schemes since they are recommended not to be supersingular or anomalous. Supersingular curves are those that have a cardinal $\#E(F_p)$ equal to *p+1*. Then, a MOV [33] attack could be done, reducing the group where the discrete logarithm problem has to be solved to break the cryptosystem. The anomalous curves have a cardinal $\#E(F_p)$ equal to *p*. Although they are resistant to the MOV attack, a polynomial algorithm to solve the discrete logarithm problem over their field could be designed.

Unless there is an assurance that domain parameters were generated by a trusted party, the domain parameters must be validated before use.

The generation of domain parameters is not usually done by each participant since this involves counting the number of points on a curve $\#E$, which is time-consuming and trouble some to implement. As a result several standard bodies published domain parameters of elliptic curves for several common field sizes, as the NIST in the document [35] and the SECG, [32]. The NIST recommended elliptic curves are chosen for optimal security and implementation efficiency.

If one wants to build his own domain parameters he should select the underlying field and then use one of the following strategies to find a curve with appropriate (i.e. near prime) number of points using one of the following methods [36]:

- Select a random curve and use a general point-counting algorithm, for example, Schoof's algorithm or Schoof-Elkies-Atkin algorithm.

- Select a random curve from a family which allows easy calculation of the number of points (eg Koblitz curves), or

- Select the number of points and generate a curve with this number of points using complex multiplication technique.

4.5.1.3. Keys

If we want to use the ECC in a public key cryptosystem, we need a public and a private key. Since by the ECDL problem it is infeasible calculate an integer *k* such that $Q = k \cdot P$ knowing *Q* and *P*, *k* is used to be the secret key while *Q* is the public key and *P* is a generator point (commonly called *G*) such that the multiplication of any integer (in the *n* range) by this point *G* results in a point belonging to the specified EC.

4.5.1.4. ECC algorithms

The EC algorithms are specified in [37]. Most of the EC cryptographic schemes are related to the discrete logarithm schemes which were originally formulated for usual modular arithmetic:

- The Elliptic Curve Diffie-Hellman key agreement scheme is based on the Difffie-Hellman scheme.

- The Elliptic Curve Digital Signature Algorithm is based on the Digital Signature Algorithm.

- The ECMQV key agreement scheme is based on the MQV key agreement scheme.

Not all the DLP schemes should be ported to the elliptic curve domain. For example, the well known ElGamal encryption scheme was never standardized by official bodies and should not be directly used over an elliptic curve (the standard encryption scheme for ECC is called Elliptic Curve Integrated Encryption Scheme). The main reason is that although it is straightforward to convert an arbitrary message to an integer modulo *p*, it is not that simple to convert a bitstring to a point of a curve.

Due to the operation complexity, the most extended applications for the ECC are the digital signature ECDSA and the key exchange protocol ECDH. In these algorithms public key cryptography is used creating a private key as a random integer and a public key as a point of the curve that is the multiplication of the private key by a generator. Instead of operating with these keys, a subkey derived from the public key (a coordinate of a point) is used to avoid the elliptic curve point operation and the conversion of the message to a point.

The main concept of these schemes is:

- *A* generates a random number k_a that is his secret key.

- *A* obtains his public key as $Q_a = k_a \cdot G$.

- *B* does the same for k_b and Q_b.

- *A* can generate a new point that only *A* and *B* will know as $P_a = k_a \cdot Q_b$.

- *B* can generate the same point $P_b = k_b \cdot Q_a = k_b \cdot k_a \cdot G = k_a \cdot Q_b = P_a$.

- Using this shared secret a message can be encrypted.

The only standardised algorithms are the named above for key agreement and for digital signature. At the RSA Conference 2005, the NSA announced Suite B [38] which exclusively uses

ECC for these uses. The suite is intended to protect both classified and unclassified national security systems and information.

In [28] these two standardised algorithms are detailed:

ECDSA: Elliptic Curve Digital Signature Algorithm

For signing a message *m* by sender *A*, using *A*'s private key k_a:

- Calculate $e = H(m)$, where *H* is a cryptographic hash function.
- Select a random integer *d* from $\{1, n-1\}$.
- Calculate $r = x_1 \; mod \; n$, where $(x_1, y_1) = d \cdot G$. If $r = 0$, go to step 2.
- Calculate $s = d^{-1} \cdot (e + k_a \cdot r) \; mod \; n$. If $s = 0$, go to step 2.
- The signature is the pair (r, s).

For *B* to authenticate *A*'s signature, *B* must know *A*'s public key Q_a:

- Verify that *r* and *s* are integers in $\{1, n-1\}$. If not, the signature is invalid.
- Calculate $e = H(m)$.
- Calculate $w = s^{-1} \; mod \; n$.
- Calculate $u_1 = e \cdot w \; mod \; n$ and $u_2 = r \cdot w \; mod \; n$.
- Calculate $(x_1, y_1) = u_1 \cdot G + u_2 \cdot Q_a$.
- The signature is valid if $x_1 = r \; mod \; n$, invalid otherwise.

ECDH: Elliptic Curve Diffie Hellman

ECDH is a key agreement protocol that allows two parties to establish a shared secret key that can be used for private key algoritms.

Let (k_a, Q_a) be the private key-public key pair of *A* and (k_b, Q_b) *B*'s key pair.

- *A* computes $P = \left(x_p, y_p\right) = k_a \cdot Q_b$.
- *B* computes $L = (x_l, y_l) = k_b \cdot Q_a$.
- Since $k_a \cdot Q_b = k_a \cdot k_b \cdot G = k_b \cdot k_a \cdot G = k_b \cdot Q_a$, *P=L* and hence $x_p = x_l$.
- The shared secret is x_p.

4.5.2. EC Homomorphic Cryptosystems

4.5.2.1. EC ElGamal Cryptosystem

As it has been explained before, due that we want to use *Integrity Proofs* in order to verify the correct mixing process, we need an homomorphic cryptosystem. The use of an EC ElGamal cryptosystem for message/vote encryption has been studied from [28][31] and exposed here.

The importance of considering the ElGamal cryptosystem over a group of points of an elliptic curve relies in that the algorithms that solve the PLD over $E(F_p)$ are harder than in F_p *, so the key size can be much smaller (160 bits versus 1024).

In order to configure such a cryptosystem the domain parameters *(p, a, b, G, n, h)* are specified, being $k \in \{1, n-1\}$ the receiver private key and $Q=k \cdot G$ the public key, where G is the point generator of the elliptic curve.

Then, the message that needs to be encrypted has to be converted into a point of the elliptic curve. The message is first converted to a natural number $(0<m<p)$, and then is usually identified with the abscissa of a point M of the curve $E(F_p)$ defined as

$$y^2=x^3+a \cdot x+b$$

Then:

$$y^2=x^3+a \cdot x+b \rightarrow y^2=m^3+a \cdot m+b \rightarrow M=(m,y)$$

If the result of $m^3+a \cdot m+b$ is not quadratic, we can try with $m+1$.

The encryption and decryption processes are explained here:

EC ElGamal encryption

- The message to be encrypted is represented as a point of the curve M.
- Choose a random integer $r \in \{1, n-1\}$.
- Calculate the elliptic curve points $C_1=r \cdot G$ and $C_2=M+r \cdot Q$.
- Send C_1, C_2.

EC ElGamal decryption

- Calculate the point $k \cdot C_1=k \cdot r \cdot G=r \cdot Q$.
- Calculate the point $M= C_2-r \cdot Q$.
- Obtain m from M.

4.5.2.2. *EC Paillier-Galbraith cryptosystem*

Paillier designed an homomorphic cryptosystem over a ring Z_n^2, $n=p \cdot q$, based on the composite residuosity problem [39].

An Elliptic Curve approach was done by S. Galbraith in [40] considering an integer $n=p \cdot q$, being p and q primes, and a curve E over the ring Z_n^2 for the scheme configuration.

One of the advantages of this scheme is that the message to encrypt m does not have to be converted to a point of the curve.

The private key in this cryptosystem is

$$k=lcm(\#E(F_p), \#E(F_q))$$

So, in order to calculate k we need to know the cardinals of the curve E over F_p and F_q.

The public key is a point Q in $\#E(Z_n^2)$ such that

$$k \cdot Q=[0, 1, 0].$$

To calculate Q we choose a random point Q', being $Q=n \cdot Q'$.

EC Paillier-Galbraith encryption

- A message m is represented as a point $P_m=[m \cdot n,1,0]$ of $E(Z_n^2)$.
- A random integer r is chosen in Z_n.
- Calculate the point $C=r \cdot Q+P_m$ in $E(Z_n^2)$.
- Send C.

EC Paillier-Galbraith decryption

- Calculate the point $k \cdot C=k \cdot(r \cdot Q+P_m)=(0,1,0)+(k \cdot m \cdot n,1,0)=(k \cdot m \cdot n,1,0)$ in $E(Z_n^2)$.
- Obtain the first coordinate $x=k \cdot m \cdot n$ of the point $k \cdot C$.
- Calculate $y=(x/n)$ in Z and the product $m=y \cdot d^{-1}$ in Z_n.
- Return m.

Using this cryptosystem, the message m is encrypted in an easier way than in ElGamal cryptosystem, due to the fact that the message is inserted directly in the point coordinates and not converted into a point. However, this elliptic curve approach, although being efficient, is not semantically secure.

A cryptosystem is semantically secure if, given two plaintexts and the ciphertext of one of them, the attacker can not decide which message is the ciphertext from with a probability higher than *1/2*.

In [41] a semantically secure cryptosystem has been developed from a generalization of the Paillier-Galbraith cryptosystem, but in this improvement the cryptosystem loses its homomorphic properties.

4.5.3. Use of the EC ElGamal cryptosystem in homomorphic schemes. Proposal for a mixing process

The main point of this cryptosystem for our purposes (achieve an homomorphic encryption that allows us to calculate the *Integrity Proofs*) relies in that we can use the additive homomorphic properties of this ElGamal scheme adaptation for elliptic curves in such a way that we can calculate the addition of the encrypted messages (*Input Integrity Proof*) and decrypt it obtaining the addition of the plaintexts. Then, if we calculate the Output *Integrity Proof* as the addition of the output plaintexts, we can compare the decrypted *Input Integrity Proof* with the *Output* one, following the original design for universal verifiability. The detailed procedure is represented in the figure 4.36:

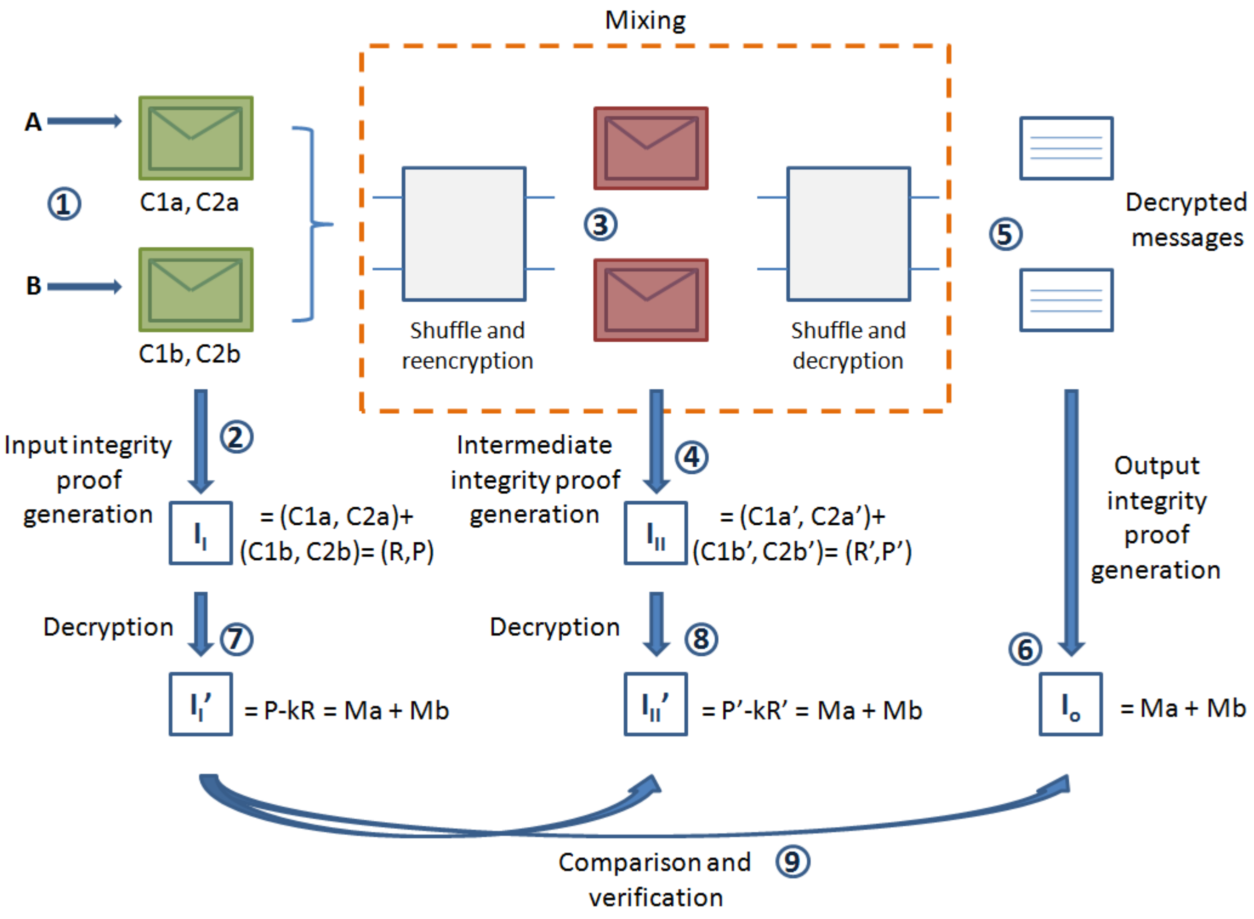

Mixing
A
B
①
C1a, C2a
C1b, C2b
Shuffle and reencryption
③
Shuffle and decryption
⑤
Decrypted messages
Input integrity proof generation
②
I_I = (C1a, C2a)+ (C1b, C2b)= (R,P)
Intermediate integrity proof generation
④
I_{II} = (C1a', C2a')+ (C1b', C2b')= (R',P')
Output integrity proof generation
Decryption
⑦
I_I' = P-kR = Ma + Mb
Decryption
⑧
I_{II}' = P'-kR' = Ma + Mb
⑥
I_o = Ma + Mb
Comparison and verification
⑨

Fig. 4 - 36 *Integrity Proof* generation using an EC cryptosystem

- Suppose that we have two parties, *A* and *B* that send each one an encrypted message to a third party with a secret key *k* and a public key *Q* as:

  $A \rightarrow (C_{1a}, C_{2a}) = (r_a \cdot G, M_a + r_a \cdot Q)$

  $B \rightarrow (C_{1b}, C_{2b}) = (r_b \cdot G, M_b + r_b \cdot Q)$

- Then, if we add both ciphertexts to calculate the Input *Integrity Proof*

  $(C_{1a}, C_{2a}) + (C_{1b}, C_{2b}) = (r_a \cdot G + r_b \cdot G, M_a + M_b + r_a \cdot Q + r_b \cdot Q) = (R, P)$

- We can decrypt the addition result and obtain the addition of both messages as:

  $M_a + M_b = P - k \cdot R = M_a + M_b + r_a \cdot Q + r_b \cdot Q - k(r_a \cdot G + r_b \cdot G) =$

  $= M_a + M_b + r_a \cdot Q + r_b \cdot Q - (r_a \cdot Q + r_b \cdot Q)$

- If we had a mixnet with re-encryption where

  $(C_{1a}', C_{2a}') = ((r_{1a} + r_{2a}) \cdot G, M_a + (r_{1a} + r_{2a}) \cdot Q)$

  $(C_{1b}', C_{2b}') = ((r_{1b} + r_{2b}) \cdot G, M_b + (r_{1b} + r_{2b}) \cdot Q)$

- We could calculate the Intermediate *Integrity Proof*, and decrypt it to recover the addition of the two messages too:

$(C_{1a}', C_{2a}') + (C_{1b}', C_{2b}') = ((r_{1a} + r_{2a}) \cdot G + (r_{1b} + r_{2b}) \cdot G, M_a + M_b + (r_{1a} + r_{2a}) \cdot Q + (r_{1b} + r_{2b}) \cdot Q) = (R', P') \rightarrow M_a + M_b = P' - k \cdot R'$

- The plaintexts in the output are operated to calculate the Output *Integrity Proof* as:

  $M_a + M_b$

- Finally, the *Integrity Proofs* can be compared to verify the correct behaviour of the mixnet.

  $I_I' = I_{II}' = I_O$

4.5.3.1. Message conversion into a point

One issue when we use the EC ElGamal encryption scheme is how to convert a message *m* to a point *M* of an elliptic curve.

A usual option [42] is, as it has been explained before, to insert the variable *m* (the message converted to an integer $<p$) in the elliptic curve equation as the *x* coordinate, being $M = (m, \sqrt{m^3 + a \cdot m + b}) = (x, y)$.

The main problem is that, if we have part of the message content on both coordinates, once that two points are added $M_a + M_b$, it is infeasible to obtain the addition of the messages as integers. As we can see in the point addition section of [28]:

Point addition:

- Consider two distinct points J and K such that $J=(x_j, y_j)$ and $K=(x_k, y_k)$.
- Let $L=J+K$ where $L=(x_l, y_l)$, then:
  $x_l = s^2 + s + x_j + x_k + a$
  $y_l = s(x_j + x_l) + x_l + y_j$
  $s = (y_j + y_k)/(x_j + x_k)$, s is the slope of the line through J and K.
- If $K=-J$, i.e. $K=(x_j, x_j + y_j)$ then $K+J=O$, where O is the point at infinity.
- If $K=J$ then $J+K=2 \cdot J$, and point doubling equations are used.

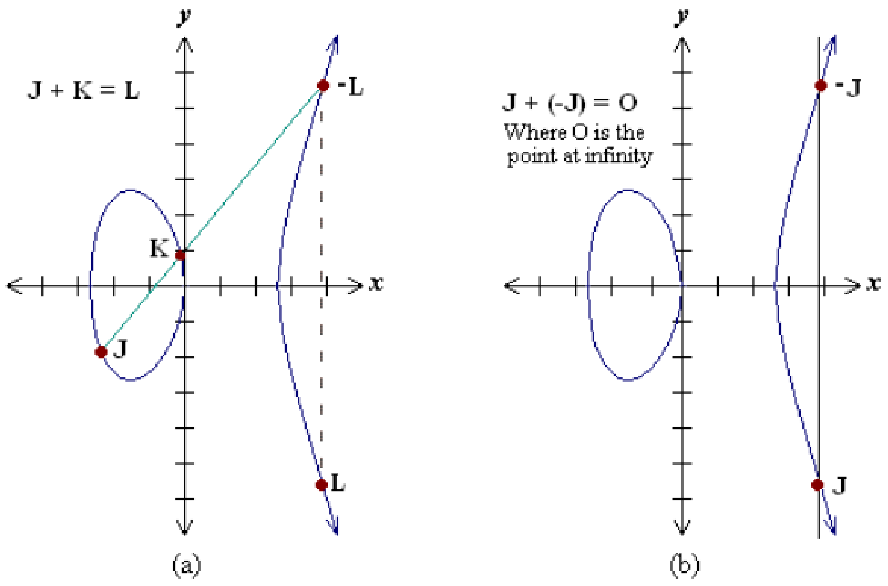


Fig. 4 - 37 Arithmetic operations in Elliptic Curves

This is not a big deal if we just want to compare the *Integrity Proofs* in the input and the output of the mix-nodes, but if we would like to have the chance of performing an homomorphic tally this cryptosystem could not be used. Otherwise, the conversion of a message into a point of the curve can difficult the posterior information extraction when the decryption is done, so we should evaluate which is the best and most efficient method.

Another option to convert a message into a point could be calculating $M=m \cdot G$. The problem is that then we would have to face the discrete logarithm problem in order to obtain m from M. There is an example in [43] where the messages are converted into points following this procedure. After all the process the final system recovers them by brute force. The system can recover the messages due to the fact that they are small integers, and then the ECDLP is not so hard to solve, but it is not our case.

Finally, the option that has been considered when converting the message is to insert it in a point with the structure $[m \cdot n{:}1{:}0]$ in an elliptic curve E over the ring Z_n, as we have observed in [36].

4.5.3.2. Use of projective coordinates

The elliptic curves over rings Z_n $(E_n(a,\ b))$, where $n=p \cdot q$, being p and q primes, are used in factorization problem based cryptosystems. The addition operation with points of an elliptic curve over a finite field can be defined for points of a curve E over a ring Z_n. However, due that there are elements in Z_n that have not inverse, the point addition in $E(Z_n)$ is not always well defined by the conventional analytic expressions.

In order to mitigate this problem, an elliptic curve defined over the projective plane $P^2(Z_n)$ can be also considered, formed by the points $(x{:}\ y{:}\ z)$ that satisfy the equation $y^2 \cdot z=x^3+a \cdot x \cdot z^2+b \cdot z^3$ in Z_n, which has semi-infinite points $(x{:}y{:}z)$ such that $lcm(z,n)$ is p or q, as well as an infinite point $O=(0{:}1{:}0)$ and affine points $(x{:}y{:}1)$.

This definition can be extended to the ring $Z_n{}^2$, the one used in the Paillier-Galbraith scheme. The natural map is $E_n{}^2(a,b) \rightarrow E_n(a,b)$.

$E_n{}^2(a,b)$ can be seen as a group isomorphic to $E_p{}^2(a,b) \times E_q{}^2(a,b)$.

Points in these curves can be classified in three types:

- Points at infinity $O_k=(k \cdot n{:}1{:}0)$, $k \in Z_n$.
- Affine points: $(x,y)=(x{:}y{:}1) \in E_n{}^2(a,b)$.
- Semi-infinite points: $(x{:}y{:}z) \in E_n{}^2(a,b)$, with $gcd(z,n)=p$ or q.

Due that the semi-infinite points provide a factorization of n they are not recommended for cryptographic uses.

The most important property for our purposes is that, given two messages m_a and $m_b \in Z_n$, the addition operation of their equivalent points is:

$[m_a \cdot n{:}1{:}0]$ and $[m_b \cdot n{:}1{:}0]$ is $P_{ma}+P_{mb}=[(m_a+m_b) \cdot n{:}1{:}0]=P_{ma+mb}$.

Instead of using a curve $E(Z_{n^2})$ as in this example, we are going to use a curve in $E\left(Z_{p^2}\right)$ in order to use the ElGamal cryptosystem modulo to define the curve.

4.5.3.3. Application of the projective coordinates to the EC-ElGamal encryption scheme

Due to the properties of the affine and the infinite point addition [44] explained below, the messages should be represented by infinite points.

Then, since we define the point addition operation as

$(m \cdot p{:}1{:}0)+(m' \cdot p{:}1{:}0)=((m+m') \cdot p{:}1{:}0)$

we can obtain the addition of the messages from the result of this operation.

The $r \cdot G$ and $r \cdot Q$ points used during the encryption process will be considered affine points.

As it is described in [44], the point operation in elliptic curves over rings $Z_p{}^2$ has the following properties:

- The addition of two points in the infinite result in a point in the infinite.

- The addition of two affine points can result in an affine point, an infinite point, or another type of point, depending on the point architecture.

- The addition of an affine point with an infinite point results in an affine point.

- If we multiply an affine point as $(x:k \cdot p:1)$ by an odd integer the result is a point in the infinite. Otherwise, the result is an affine point.

In the ElGamal cryptosystem, we define the points $r \cdot G$ and $r \cdot Q$ as affine points at first.

During the calculation of the *Integrity Proofs* or the re-encryption process these points are added to others. Therefore, these affine points can be converted in infinite points through all these operations.

The question is: could an attacker extract any information of this?

Let's see what happens when we add two encrypted messages (i.e. when we calculate the *Integrity Proofs*):

Being the encrypted messages

$$A \rightarrow (C_{1a}, C_{2a}) = (r_a \cdot G, M_a + r_a \cdot Q)$$

$$B \rightarrow (C_{1b}, C_{2b}) = (r_b \cdot G, M_b + r_b \cdot Q)$$

The attacker sees the result of the operation

$$(C_{1a}, C_{2a})+(C_{1b}, C_{2b})=(r_a \cdot G+r_b \cdot G , M_a+M_b+r_a \cdot Q+r_b \cdot Q)=(R, P)$$

And can extract some conclusions from the point operation properties presented above:

- If the point P is at the infinite, it would mean that so does $k \cdot R=(r_a \cdot Q+r_b \cdot Q)$.

  Since

  $$P=M_a+M_b+r_a \cdot Q+r_b \cdot Q,$$

  and M_a and M_b are both infinite points.

- If the point R is an affine point, and $k \cdot R$ is an infinite one, k should be odd.

- Hence, if an attacker detects that *P* is an infinite point, he can just reduce the possible values of the secret key *k* to the half. In case of having a secret key *k* of 512 bits, the possible values would be reduced from 2^{512} to 2^{511}.

4.5.3.4. How to solve the DLP using projective coordinates

While we were working on how to apply the projective coordinates to the EC-ElGamal cryptosystem, we realised that the Discrete Logarithm Problem could be easily solved working in $Z_p^{\,2}$.

The number of rational points of a curve $E(Z_p)$ is $\#E(Z_p)$.

When we work in a projective plane and use the curve $E(Z_p^{\,2})$,

$\#E(Z_p^{\,2})=p\cdot \#E(Z_p)$,

where $\#E(Z_p)= l$ and can be calculated using the Schoof algorithm [45]. So $\#E(Z_p^{\,2})=p\cdot l$.

Being R a point belonging to $E(Z_p^{\,2})$ where $R \bmod p \in E(Z_p)$,

$$l \cdot p \cdot R = (k \cdot p\colon 1\colon 0) \leftarrow \text{a point in the infinite.}$$

In the section 4.5.3.3. a message *m* is encrypted using a private key *d* and a public key $Q = d \cdot G$, where *G* is the generator:

$$(r \cdot G, r \cdot Q + m \cdot p\colon 1\colon 0)$$

If we calculate:

- $l \cdot p \cdot Q = l \cdot d \cdot G = (\delta \cdot p\colon 1\colon 0)$
- $l \cdot p \cdot G = (k \cdot p\colon 1\colon 0)$
- $\frac{\delta \cdot p}{k \cdot p} = d.$

We could recover the secret key *d*.

Our conclusion is that the projective coordinates in Z_{p^2} cannot be used in combination with cryptosystems like ElGamal, since the DLP is easy to solve here.

Since we cannot use this system to insert a message into a point of an elliptic curve and the other systems we have found do not work for systems where we want to recover the content of both messages from their operation (due to that they have to be converted to points of the Elliptic Curve), we have decided to leave it for future work.

5. Zero Knowledge Proofs in ElGamal cryptosystems

The Zero-Knowledge Proofs are advanced cryptographic mechanisms that are commonly used in e-Voting.

There is a well-known story presenting some of the ideas of Zero-Knowledge Proofs, first published in [46]. Commonly, the two parties in a zero-knowledge protocol are named as Peggy (the prover of the statement) and Victor (the verifier of the statement).

In this story, Peggy has uncovered the secret word used to open a magic door in a cave. The cave is shaped like a circle, with the entrance on one side and the magic door blocking the opposite side. Victor says he'll pay her for the secret, but not until he's sure that she really knows it. Peggy says she'll tell him the secret, but not until she receives the money. They devise a scheme by which Peggy can prove than she knows the word without telling it to Victor.

First, Victor waits outside the cave as Peggy goes in. The left and right paths from the entrance are labelled as A and B. She randomly takes either path A or B. Then, Victor enters the cave and shouts the name of the path he wants her to use to return, either A or B, chosen at random. Providing she really does know the magic word, she opens de door, if necessary, and returns along the desired path. The process is depicted in figure 5.1. Note that Victor does not know which path she has gone down.

However, suppose she did not know the word. Then, she would only be able to return by the named path if Victor were to give the name of the same path that she had entered by. Since Victor would choose A or B at random, she would have a 50% chance of guessing it correctly. If they were to repeat this trick many times, say 20 times in a row, her chance of successfully anticipating all of Victor's requests would become vanishingly small.

Thus, if Peggy reliably appears at the exit Victor names, he can conclude that she is very likely to know the secret word.

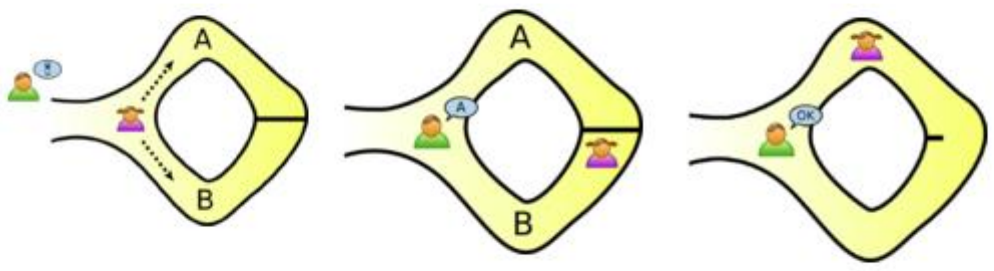


Fig. 5 - 1 Illustration of the cave with a magic door

A Zero-Knowledge Proof must satisfy three properties:

- **Completeness:** if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.

- **Soundness:** if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.

- **Zero-Knowledge:** if the statement is true, no cheating verifier learns anything other than this fact.

  Computational vs. perfect zero-knowledge: A protocol is computationally zero-knowledge if an observer restricted to probabilistic polynomial-time tests cannot distinguish real from simulated transcripts. For perfect zero-knowledge, the probability distributions of the transcripts must be identical.

ZKP are not proofs in the mathematical sense of the term because there is some small probability, the *soundness error*, that a cheating prover will be able to convince the verifier of a false statement. They are probabilistic rather than deterministic. However, there are techniques to decrease the soundness error to negligibly small values.

Here we are going to explain some of the ZKP [13][47][48][49] that can be used in the ElGamal cryptosystem, and how these can be adapted to be used in the new designed proposal.

At first each ZKP is explained to be used in an ElGamal encryption scheme defined in the group Zp, where q is a prime number, and p a safe prime $p=2q+1$. The cryptosystem uses a generator g of the q-order subgroup Gq of Zp^*. There secret key x is generated as random number in Zq, and the public key is $h=g^x mod\ p$.

The new cryptosystem proposal is defined with the same parameters than the conventional ElGamal encryption system, and 3 secret (x_1, x_2, x_3) and public keys are used:

$$h_1=g^{x1} mod\ p$$
$$h_2=g^{x2} mod\ p$$
$$h_3=g^{x3} mod\ p$$

All the modular operations are omitted from now.

5.1. Schnorr Identification Protocol

An Identification Protocol [17] is a technique designed to allow one party, the verifier, to gain assurances that the identity of another, the claimant, is as declared, thereby preventing impersonation.

This is a Zero-Knowledge identification protocol, that is to say, a cheater verifier cannot learn anything from the proof that lets him impersonate the prover identity.

The security of this protocol is based on the intractability of the discrete logarithm problem. The basic idea is that A proves knowledge of a secret x (without revealing it) in a time-variant manner (depending on a challenge *e*), identifying A through the association of *x* with the public key *h* via A's authenticated certificate.

5.1.1. Protocol:

A identifies itself to B as follows:

- A chooses a random *e* (the commitment), $1 \leq e \leq q-1$, computes the witness $w = g^e$, and sends that witness and its public certificate to B.
- B authenticates A's public key *h* by verifying the certificate, then sends to A a never previously used random *c* (the challenge), $1 \leq c \leq 2^t$, where 2^t is a pre-defined security level.
- A checks that $1 \leq c \leq 2^t$ and sends B the response *s=xc+e*.
- B verifies that $g^s = w \cdot h^c$

5.1.2. Schnorr Signature

The Schnorr Signature is a non-interactive protocol. It is basically equal to the Schnorr Identification Protocol, but replacing the challenge by a hash on the prover commitment.

Being a message m encrypted as $(m \cdot h^r, g^r) = (a, b)$, the signature over the encrypted message is performed as:

- *M=H(a|b|h|g)*
- Choose a random *e* (the commitment), $1 \leq e \leq q-1$
- Compute the witness $w = g^e$
- Calculate *c=H(M|w)* and *s=(e-rc)*

The signature is *(c,s)*

The signature can be verified as:

- Compute $rv = g^s \cdot b^c = g^{e-rc} \cdot b^c = g^{e-rc} \cdot g^{rc} = g^e = w'$
- Compute *c'=H(M|w')*
- Verify that *c'=c*.

5.2. Proof of Discrete-Log equality - Chaum-Pedersen Protocol

In [51], a method is presented that permits a prover to prove in zero knowledge that, given a tuple *(g,u,h,v)*, he knows a secret value *x* satisfying $x = \log_g h = \log_u v$, also known as the proof of discrete-log equality. This proof, initially interactive, can be made non-interactive using the procedure described in the Fiat-Shamir scheme [50].

Being *m* the decryption of a ciphertext $c = (m \cdot h^r, g^r)$, the tuple (g, u, h, v) can be formed, where *g* and *h* are public parameters of the ElGamal cryptosystem, $u = g^r$ and $v = \frac{m \cdot h^r}{m} = h^r$.

Then, the prover can prove in zero knowledge that he knows the secret key *x* used to decrypt *c* as *m*:

- The prover chooses a random *e* (the commitment), $1 \leq e \leq q - 1$ and computes the witness $(a, b) = (g^e, u^e)$.

- Then, he computes *d=H(h|g|a|b|u|v).*

- Finally, he calculates *s=xd+e*, and sends *(a,b,s)* to the verifier.

- The verifier computes *d=H(h|g|a|b|u|v)*, and checks that

$$g^s = a \cdot h^d$$
$$u^s = b \cdot v^d$$

In the cryptosystem presented in the Proposal 6 several parameters are encrypted using a conventional ElGamal cryptosystem. Thus, being *c* the encryption of a message *m* divided in *n* blocks,

$c = (\{a_i \dots a_n\} \cdot {h_1}^{r_1}, g^{r_1}, y \cdot {h_2}^{r_1}, r_2 \cdot {h_3}^{r_1})$,
(note that we consider that the message has been re-encrypted)

where

$a_i = [m_i \cdot H(y, i) || H(m_i \cdot H(y, i))] \cdot {r_2}^{H(i)}$,

the parameter r_2 is encrypted as

$(g^{r_1}, r_2 \cdot {h_3}^{r_1})$,

and a Chaum-Pedersen proof can be done in order to proof the correct decryption of r_2.

For each block, the parameter a_i is encrypted as

$(g^{r_1}, a_i \cdot {h_1}^{r_1})$,

and again the correct decryption of a_i can be proved using a Chaum-Pedersen proof.

The parameter y is encrypted as

$(g^{r_1}, y \cdot {h_2}^{r_1})$,

And another Chaum-Pedersen proof can be done to proof the correct decryption of y.

Thus, proving the correct decryption of r_2, a_i and y, the correct decryption of each message block m_i can be proved.

The prover can prove in zero knowledge that he knows the secret key x_i used to decrypt each parameter:

- First, the tuple $(g, u, h_1, h_2, h_3, v_1, v_2, v_3)$ can be constructed, where *g* and *h* are public parameters of the ElGamal cryptosystem, $u = g^{r_1}$, $v_1 = \frac{a_i \cdot {h_1}^{r_1}}{a_i} = {h_1}^{r_1}$, for all the *n* blocks, $v_2 = \frac{y \cdot {h_2}^{r_1}}{y} = {h_2}^{r_1}$, and $v_3 = \frac{r_2 \cdot {h_3}^{r_1}}{r_2} = {h_3}^{r_1}$.

- The prover chooses a random *e* (the commitment), $1 \leq e \leq q-1$ and computes the witness $(a, b) = (g^e, u^e)$.

- Then, he computes *d=H(*$h_1/h_2/h_3/g/a/b/u/v_1/v_2/v_3$*)*.

- Finally, he calculates $s_1 = x_1 \cdot d + e$, $s_2 = x_2 \cdot d + e$, $s_3 = x_3 \cdot d + e$, and sends *(a, b,* s_1*,* s_2*,* s_3*)* to the verifier.

- The verifier computes *d=H(*$h_1/h_2/h_3/g/a/b/u/v_1/v_2/v_3$*)*, and checks that

$$g^{s_1} = a \cdot {h_1}^d$$
$$u^{s_1} = b \cdot {v_1}^d$$

$$g^{s_2} = a \cdot {h_2}^d$$
$$u^{s_2} = b \cdot {v_2}^d$$

$$g^{s_3} = a \cdot {h_3}^d$$
$$u^{s_3} = b \cdot {v_3}^d$$

- Finally, with the verification of the parameters r_2, a_i and y, the correct decryption of each message block m_i can be proved.

5.3. Proof of Plaintext Knowledge

Since ElGamal is a malleable cryptosystem, one can forge encrypted messages from one already encrypted without knowing its content. For example:

$$c = (m \cdot h^r, g^r)$$
$$c' = (2,1) \cdot c = (2 \cdot m \cdot h^r, g^r)$$

If we are using a mixnet in order to protect voters' privacy, this malleability can endanger the process. For example, if an attacker wants to know what a voter has voted, he could forge an encrypted vote multiplying the one cast by the voter by a known factor. Then, when the votes are decrypted, the attacker could search for the pair of votes that maintain the same algebraic relationship that the one cast by the voter and the one forged by the attacker and find how the voter has voted. This is called a message-related attack.

This is the reason why the encrypted messages cast by the voters should be non-malleable. Informally, non-malleability means that it is infeasible, given a ciphertext, to create a different ciphertext such that their plaintexts are related. The non-malleability under message-related attacks is achieved using non-malleable non-interactive zero knowledge proofs of knowledge of the plaintext. Non-malleability is an extension of semantic security since it considers security and self-protection of senders in the context of a network of users, and not simply between one sender and one receiver (in the case of a man-in-the-middle attack).

This ZKP of plaintext knowledge is typically given by a digital signature. In [49], it is explained how a Schnorr signature is forged from the message encryption and attached to it (explained in Section 5.1.1) to let the receiver verify that the message was originally properly encrypted.

When we use the cryptosystem presented in the Proposal 6, although it is non-malleable due to the hash inside the encryption, this can only be verified at the decryption time. If we want to detect a message-related attack before the decryption process (usually between the vote submission and the mixing steps) we have to perform a Schnorr signature over the encryption.

Being *c* an encrypted message *m* divided in *n* blocks:

$$c = (\{a_i \dots a_n\} \cdot {h_1}^{r_1}, g^{r_1}, y \cdot {h_2}^{r_1}, {h_3}^{r_1}) = (\mathrm{a}, \mathrm{b}, \mathrm{d}, \mathrm{e}),$$

where

$$a_i = m_i \cdot H(y,i) || H(m_i \cdot H(y,i)),$$

the signature over the encrypted message is performed following the next steps:

- Calculate $M=H(h_1/h_2/h_3/g/a/b/d/e)$.
- Choose a random *z* (the commitment), $1 \leq z \leq q-1$
- Compute the witness $w = g^z$
- Calculate $v=H(M/w)$ and $s=(z-r_1v)$

The signature is *(v,s)*

The signature can be verified as:

- Compute $rv = g^s \cdot b^v = g^{z-r_1 v} \cdot b^v = g^{z-r_1 v} \cdot g^{r_1 v} = g^z = w'$
- Compute $v'=H(M/w')$
- Verify that $v'=v$.

5.4. Proof of Plaintext Equivalence

This proof [13] can be constructed exploiting the homomorphism property of the ElGamal cipher. It is based in the Schnorr Identification Protocol: used in a proper manner this proof can be used to prove that one entity knows the re-encryption factor used to re-encrypt an encrypted message without disclosing it.

Being an encrypted message
$c = (m \cdot h^r, g^r)$

And its re-encryption
$c' = c \cdot \left(h^{r'}, g^{r'}\right) = (m \cdot h^{r+r'}, g^{r+r'})$

The entity that executes the re-encryption process can proof that *c* and *c′* have the same content by:

- Calculating $c'/c = \left(1 \cdot h^{r'}, g^{r'}\right)$
- Being $h' = h^{r'}$, and $g' = h$, the prover can demonstrate that he knows the re-encryption factor *r′* as if it was the private key in an ElGamal cryptosystem, where $h' = g'^{r'}$ using the Schnorr Identification Protocol:
    - The prover chooses a random *e* (the commitment), $1 \leq e \leq q - 1$, and computes the witness $w = g'^{e}$.
    - The prover calculates a challenge (to make the protocol non-interactive) as *d=H(w|h′|g′|h|g)*.
    - Finally, the prover calculates *s=r′d+e,* and sends *[w,s]* to the verifier.
- The verifier calculates h' and g' from the messages *c* and *c′*, and calculates *d*.
- Then, he verifies that $g'^{s} = w \cdot h'^{d}$

When the cryptosystem presented in the Proposal 6 is used to encrypt a message *m* divided in *n* blocks, the result is

$$c = (\{a_i \dots a_n\} \cdot {h_1}^{r_1}, g^{r_1}, y \cdot {h_2}^{r_1}, {h_3}^{r_1}) = (\mathrm{a}, \mathrm{b}, \mathrm{d}, \mathrm{e})$$

where

$$a_i = m_i \cdot H(y,i) || H\left(m_i \cdot H(y,i)\right),$$

and the re-encryption is performed as

$$c' = (\{a_i' \dots a_n'\} \cdot {h_1}^{r_1+r_1'}, g^{r_1+r_1'}, y \cdot {h_2}^{r_1+r_1'}, r_2 \cdot {h_3}^{r_1+r_1'}) = (\mathrm{a}', \mathrm{b}', \mathrm{d}', \mathrm{e}'),$$

where

$$a_i' = [m_i \cdot H(y,i) || H\left(m_i \cdot H(y,i)\right)] \cdot {r_2}^{H(i)}$$

Since in the Proposal 6 the re-encryption process includes the multiplication of the parameters *a* and *e* of the tuple by random numbers ($\{{r_2}^{H(i)}\}$, r_2), this proof does not work here at first. We could perform the proof between *b* and *d*, but not using *a* and *e* because, although they are multiplied by the same random number, this random number is exponentiated to a hash in *a*, so it seems that they cannot be related to do this proof.

An approach to the problem could be to calculate the proof between the elements *(b, b', d, d')*, and then try to perform it between the elements *(a, a', b, b', e, e')* as follows:

- Calculate *c'/c* to obtain *(a'/a,b'/b,d'/d,e'/e)* :

$$\{{r_2}^{H(i)}\} \cdot {h_1}^{r_1'}, g^{r_1'}, {h_2}^{r_1'}, r_2 \cdot {h_3}^{r_1'}$$

- Proof the knowledge of r_1' using $g' = h_2$ and $h' = {h_2}^{r_1'}$, and following the standard protocol.

- Proof the knowledge of the re-encryption factor r_1' equal to the one in the step before using $g' = h_3/h_1$ and $h_i' = \{s_i\} \cdot (h_3/h_1)^{r_1'}$, where $s_i = r_2/({r_2}^{H(i)})$ for each encrypted message block a_i:

    - The prover sends the factors $t = {r_2}^{r_1'-1}$ and $k_i = {r_2}^{{H(i)}^{r_1'-1}}$, for *i=1…n*.

    - The verifier checks that $k_i = t^{H(i)}$, for each block (*i=1…n*).

    - The prover and the verifier calculate $h_i'' = h_i' \cdot \frac{t}{k_i} = (\{s_i\} \cdot (h_3/h_1))^{r_1}$.

    - Finally, the proof of knowledge of r_1' can be calculated using $g''= \{s_i\} \cdot g'$ and h_i'' for each block.

5.5. Proof of Subset Membership

Being $c = (m \cdot h^r, g^r)$ the encryption of a message *m*, this proof lets a prover (who knows *m*) convince a verifier that the ciphertext is an encryption of a value in the set $\{m_1, \dots, m_n\}$.

In electoral processes this can be useful to verify that a voter sends a ciphertext corresponding to a valid option. It is commonly used in schemes where homomorphic tally is used. For example, when exponential ElGamal encryption is used for this purpose, this kind of proof is done to ensure that the voters choose exponents equal to *0* or *1* in order to prevent one voter to modify the election result by voting more times than allowed for one candidate:

In exponential ElGamal a message is encrypted as

$c = (\lambda^{ch} \cdot h^r, g^r)$, where *ch={0,1}*.

If a voter chooses *ch=100*, the homomorphic tally could count 100 votes for the candidate for just one vote. So the candidate should proof that the encrypted ballot contains a message $m = \{\lambda^0, \lambda^1\}$ in order to prevent this attack.

For a message m_i encrypted as $c = (m_i \cdot h^r, g^r) = (y, u)$, the prover proofs that m_i is a member of the set $\{m_1, \dots, m_n\}$ following these steps:

- Calculate the pair

  $a_j = g^{e_j} \cdot u^{d_j}$
  $b_j = h^{e_j} \cdot (y/m_j)^{d_j}$

  For each *j=1..n*, being $j \neq i$, where e_j, d_j are random numbers in *Zp*.

- Calculate the pair

  $a_i = g^w$
  $b_i = h^w$

  For the encrypted message, where w is a random number in *Zp*.

- Send to the verifier $\{a_j, b_j\}$, for all the values *j=1…n*.

- The verifier sends a challenge *f* in *Zp* to the prover (it could be substituted by a hash function, following the Fiat-Shamir procedure).

- The prover calculates

  $d_i = f - \sum_{j=0;j\neq i}^{n} d_j$
  $e_i = w - r \cdot d_i$, where *r* is the random encryption factor of the ciphertext.

- Finally the prover sends to the verifier $\{e_j, d_j\}$, for all the values *j=1…n*.

- The verifier checks that

  $f = \sum_{j=0}^{n} d_j$, and that

  $a_j = g^{e_j} \cdot u^{d_j}$
  $b_j = h^{e_j} \cdot (y/m_j)^{d_j}$

  For each *j=1..n*.

Then, the verifier has verified that the ciphertext *c* encrypts a message m_i from the set $\{m_1, \dots, m_n\}$, without knowing which one.

This proof can not be performed when the cryptosystem presented in Proposal 6 is used, due that the content of each encrypted block a_i is randomized and can not be pre-calculated ($a_i = m_i \cdot H(y, i) || H(m_i \cdot H(y, i))$, where *y* is a random parameter).

A solution should be found to solve this problem and remains for future work. Since this proof is basically used in homomorphic tally systems, the fact of not being able to do it using the designed cryptosystem does not prevent from its use in a mixing system, where the correct structure of a vote can be verified at the decryption time.

5.6. Proof of Correct Decryption using Schnorr Identification Protocol

We use the Schnorr Identification Protocol again for this proof, in order to demonstrate that an entity that does a decryption process knows the secret key *x*:

Being a message *m* encrypted as

$$c = (m \cdot h^r, g^r),$$

the decryption result must be *m*.

The prover can demonstrate that knows the secret key *x* used to decrypt the ciphertext *c* as the message *m* by:

- Calculating $c/m = (h^r, g^r)$
- Being $h' = h^r$, and $g' = g^r$, the prover can demonstrate that he knows the private key *x,* where $h' = g'^x$, using the Schnorr Identification Protocol:
  - The prover chooses a random e (the commitment), $1 \leq e \leq q-1$, and computes the witness $w = g'^e$.
  - The prover calculates a challenge (to make the protocol non-interactive) as *d=H(w|h'|g'|h|g)*.
  - Finally, the prover calculates *s=xd+e,* and sends *[w,s]* to the verifier.
- The verifier calculates h' and g' from the messages *c* and *m*, and calculates *d*.
- Then, he verifies that $g'^s = w \cdot h'^d$

Although the Chaum-Pedersen Protocol is commonly used in order to verify the correct decryption of an encrypted message, this proof is more efficient, due that it requires to interchange less information between prover and verifier and less verification operations.

In the cryptosystem presented in the Proposal 6 several parameters are encrypted using a conventional ElGamal cryptosystem. Thus, being *c* an encrypted message *m* divided in *n* blocks,

$$c = (\{a_i \dots a_n\} \cdot {h_1}^{r_1}, g^{r_1}, y \cdot {h_2}^{r_1}, r_2 \cdot {h_3}^{r_1}),$$

where

$$a_i = [m_i \cdot H(y,i) || H(m_i \cdot H(y,i))] \cdot {r_2}^{H(i)},$$

the parameter r_2 is encrypted as

$$(g^{r_1}, r_2 \cdot {h_3}^{r_1}).$$

For each block, the parameter a_i is encrypted as

$(g^{r_1}, a_i \cdot {h_1}^{r_1})$,

and the parameter y is encrypted as

$(g^{r_1}, y \cdot {h_2}^{r_1})$.

The Schnorr proof can be done in order to proof the correct decryption of r_2, a_i (for each block), and y. Thus, proving the correct decryption of r_2, a_i and y, the correct decryption of each message block m_i can be proved.

The prover can prove in zero knowledge that he knows the secret key x_i used to decrypt each parameter:

- Calculating $c/[\{a_i \dots a_n\}, 1, y, r_2] = ({h_1}^{r_1}, g^{r_1}, {h_2}^{r_1}, {h_3}^{r_1})$
- Being ${h_1}' = {h_1}^{r_1}$, ${h_2}' = {h_2}^{r_1}$, ${h_3}' = {h_3}^{r_1}$ and $g' = g^{r_1}$, the prover can demonstrate that he knows the private key x_i , where ${h_i}' = g'^{x_i}$ using the Schnorr Identification Protocol:
    - The prover chooses a random z (the commitment), $1 \leq z \leq q - 1$, and computes the witness $w = g'^{z}$.
    - The prover calculates a challenge (to make the protocol non-interactive) as *v=H(w/${h_1}'$/${h_2}'$/${h_3}'$/g'/h_1/h_2/h_3/g)*.
    - Finally, the prover calculates $s_1 = x_1 \cdot v + z$, $s_2 = x_2 \cdot v + z$, $s_3 = x_3 \cdot v + z$, and sends *(w, s_1, s_2, s_3)* to the verifier.
- The verifier calculates ${h_1}' = {h_1}^{r_1}$, ${h_2}' = {h_2}^{r_1}$, ${h_3}' = {h_3}^{r_1}$ and $g' = g^{r_1}$ from the encrypted message *c* and the decrypted parameters $[\{a_i \dots a_n\}, 1, y, r_2]$ and calculates *v*.
- Then, he verifies that

$$g'^{s_1} = w \cdot {h_1}'^{v}$$
$$g'^{s_2} = w \cdot {h_2}'^{v}$$
$$g'^{s_3} = w \cdot {h_3}'^{v}$$

- Finally, with the verification of the parameters r_2, a_i and y, the correct decryption of each message block m_i can be proved.

6. Case Study: use of the proposed encryption algorithm in a Universally Verifiable Mixnet

In this chapter we are going to present how the designed cryptosystem could be used in a real voting system.

The voting system is composed by the following entities:

- Voters:

  When the electoral process starts and the voting stage is opened, the eligible voters (using a web interface) access to the election website, choose their voting options and submit them to the Voting Service.

- Voting Client:

  The Voting Client is an application running in the voter PC that does all the cryptographic operations in order to send the encrypted ballot and ZK proofs to the Voting Service.

- Electoral Board:

  The Electoral Board is formed by parties with different interests. They have the private key that can decrypt the received encrypted votes. This key is usually divided in shares using a secret sharing threshold scheme where a determined subset of members can recover the secret key from their shares.

- Voting Service:

  The Voting Service receives the encrypted votes submitted by the voters, verifies their digital signatures (each eligible voter is assumed to have public credentials to perform a digital signature) with the electoral roll and the ZK proof of plaintext knowledge of each vote.

- Mixing Service:

  The Mixing Service shuffles the votes to protect the voters' privacy before the decryption process is done.

- Counting Service

  The Counting Service receives the mixed and decrypted votes and calculates the results for each candidate.

All the entities are assumed to have public and private RSA keys in order to digitally sign their output contents. The next entity in the chain will verify the integrity of these contents before processing them.

In the figure 6.1 the relationship between these entities is presented.

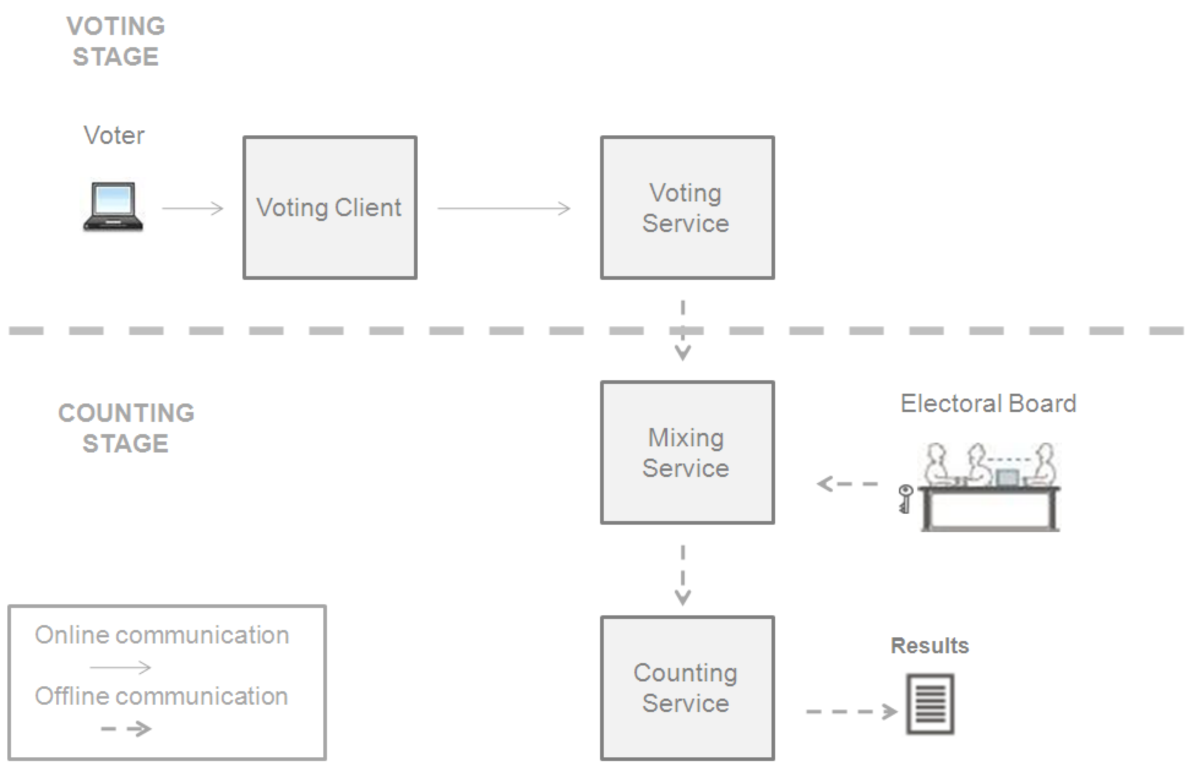


Fig. 6 - 1 Entities in the case study

6.1. Defining the cryptosystem

The votes are encrypted using the modification of the ElGamal cryptosystem defined in the Proposal 6. Here we introduce again the encryption parameters:

Being q a prime number and p a safe prime such that $p=2q+1$, the modification of the ElGamal cryptosystem is defined in the group Zp using a generator g of the q-order subgroup Gq of Zp^*.

There are 3 secret keys, x_1, x_2, x_3, generated as random numbers in Zq.

The public keys are:

$h_1=g^{x1} mod\ p$
$h_2=g^{x2} mod\ p$
$h_3=g^{x3} mod\ p$

The three secret keys are only known by the Electoral Board members. The Electoral Board could be divided in three subgroups of members, where each subgroup could possess one of the secret keys.

The three public keys are known by all the participants in the election process.

6.2. Voting Phase

6.2.1. Vote casting

In the voting phase the voter selects her choices for the current election. These choices are sent to the Voting Client, which performs the cryptographic operations needed.

Being $\{m_i\}$, $i = 1..k$ the options selected by the voter, the Voting Client performs their encryption as

$$c = (\{a_i \dots a_k\} \cdot {h_1}^{r_1}, g^{r_1}, y \cdot {h_2}^{r_1}, {h_3}^{r_1}) = (\mathrm{a}, \mathrm{b}, \mathrm{d}, \mathrm{e}),$$

where $a_i = m_i \cdot H(y, i)||H(m_i \cdot H(y, i))$, and r_1 is a random number in Z_p.

Then, a proof of plaintext knowledge is calculated performing a Schnorr signature over the encrypted options.

The Schnorr signature over the encrypted vote is performed as:

- *M=H($h_1/h_2/h_3/g/a/b/d/e$).*
- Choose a random *z* (the commitment), $1 \leq z \leq q - 1$
- Compute the witness $w = g^z$
- Calculate *v=H(M/w)* and *s=(z-r_1v)*

The signature is *(v,s)*

Finally, the encrypted vote and the ZK proof are digitally signed using the voter's private key and sent to the Voting Service:

$$Sig_V(c, [v, s])$$

6.2.2. Vote reception

When the Voting Service receives an encrypted vote, it performs two verification steps before storing the vote:

1. Verification of the digital signature: the Voting Service verifies that the public key used to verify the digital signature over the encrypted vote is from a voter in the Electoral Roll.

2. Verification of the proof of plaintext knowledge:

   The Schnorr signature *[v,s]* over the encrypted vote *c* can be verified following these steps:

   - Compute *M=H(*$h_1/h_2/h_3/g/a/b/d/e$*).*
   - Compute $rv = g^s \cdot b^v = g^{z-r_1 v} \cdot b^v = g^{z-r_1 v} \cdot g^{r_1 v} = g^z = w'$
   - Compute *v'=H(M/w')*
   - Verify that *v'=v*.

Once these two proofs are verified, the encrypted vote is stored the Voting Service.

6.3. Mixing Phase

When the voting stage is closed, all the digital ballot boxes in the Voting Servers are collected. These digital ballot boxes are signed by the corresponding Voting Server in order to ensure their integrity during the collection stage. When these digital ballot boxes are received by the Mixer, it verifies their signatures before performing the mixing process.

Once the digital ballot boxes are "opened", the digital signature over each encrypted ballot is verified. After the verification, these credentials are removed, since it would have no sense to shuffle the encrypted ballots in order to decorrelate them from the voters when they have the voters' digital signatures attached to them.

The mixing process is performed by a re-encryption mixnet. The verification of the mixnet procedure is slightly more complex that the one we took as a basis for the design of the cryptosystem. The modification is done in order to offer more protection in front of the *"300 attack"* explained in Section 4.

The verification method is based on the idea of RPC (see Section 2), but using *Integrity Proofs*. We call it *Random Group Full Checking*.

In Random Group Full Checking, the input encrypted votes of each node are divided in several groups. An *Integrity Proof* is calculated for each group of input encrypted votes and the ones belonging to this group in the output, so the process can be verified.

The next figure shows how the input and output groups are related:

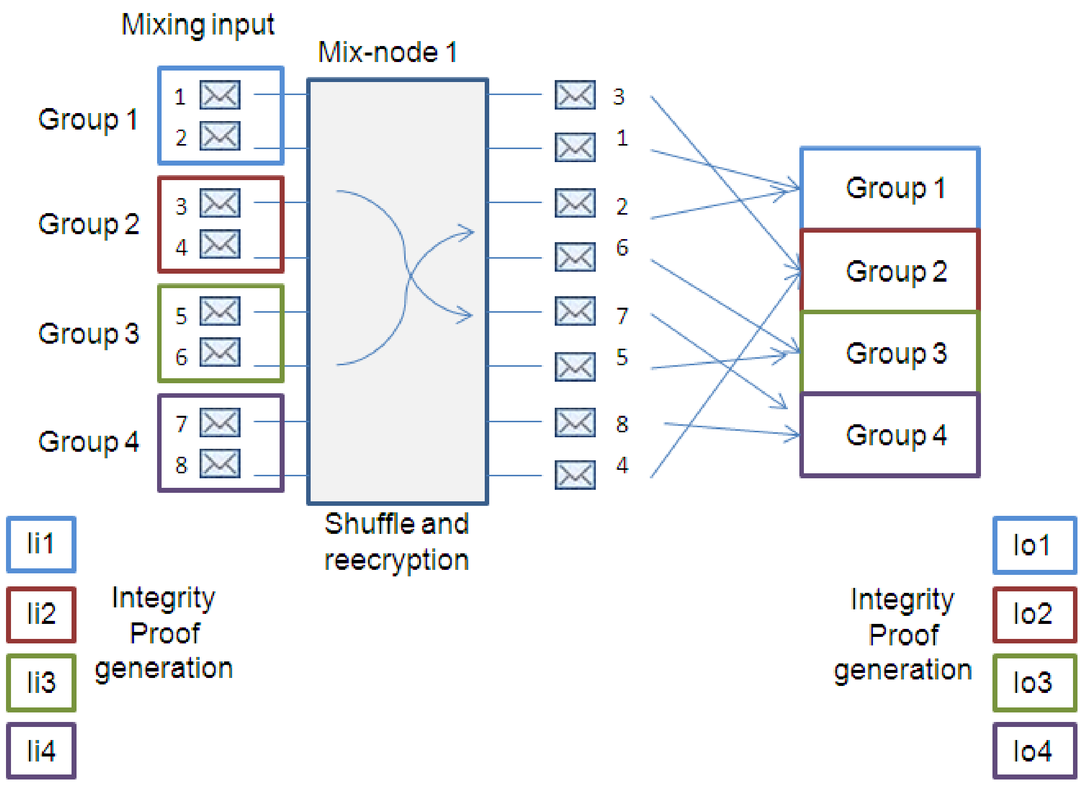


Fig. 6 - 2 Input and output groups relationship

The steps to implement the RGFC are the following:

- The mix-nodes shuffle and re-encrypt the input votes using new randomization values. In order to re-encrypt the input votes, each node performs the following operations:
  - Choose two random parameters in *Zq*, r_1' and r_2'.
  - Perform the re-encryption process as:
    - $b' = b \cdot g^{r_1'}$.
    - $d' = d \cdot {h_2}^{r_1'}$.
    - $e' = e \cdot {h_3}^{r_1'} \cdot {r_2}'$.
    - $a' = a \cdot {h_1}^{r_1'} \cdot \{{r_2}'^{H(i)}\}$,

    where the operation

    $$a_i' = [m_i \cdot H(y,i)||H(m_i \cdot H(y,i))] \cdot {h_1}^{r_1+r_1'} \cdot {r_2}'^{H(i)}$$

    is performed for each vote option.

- Once the mixing process has finished, the Electoral Board (the verifier) verifies its correct behaviour:

  - For the first node, the verifier divides at random the total of votes in groups of *n* votes. Then, it multiplies the votes contained in each group, obtaining an Input Integrity Proof:

$$I_{IIi} = \left(\prod_{j=1}^{n} a_{ij}, \prod_{j=1}^{n} g^{(r_1)_j}, \prod_{j=1}^{n} y_j \cdot h_2^{(r_1)_j}, \prod_{j=1}^{n} h_3^{(r_1)_j} \right)$$
$$= \left(\prod_{j=1}^{n} (m_{ij} \cdot H(y_j, i) || H\left(m_{ij} \cdot H(y_j, i)\right)) \cdot h_1^{(r_1)_j}, \prod_{j=1}^{n} g^{(r_1)_j}, \prod_{j=1}^{n} y_j \cdot h_2^{(r_1)_j}, \prod_{j=1}^{n} h_3^{(r_1)_j} \right)$$

  Where *j=1...n*, being n the number of votes in a group.

  - The verifier asks to the node, for each group, the destination in the outputs of its components, in order to calculate an Output Integrity Proof over the members:

$$I_{Ioi} = \left(\prod_{j=1}^{n} a_{ij}', \prod_{j=1}^{n} g^{(r_1+r_1')_j}, \prod_{j=1}^{n} y_j \cdot h_2^{(r_1+r_1')_j}, \prod_{j=1}^{n} h_3^{(r_1+r_1')_j} \cdot r_{2j}' \right)$$
$$= \left(\prod_{j=1}^{n} (m_{ij} \cdot H(y_j, i) || H\left(m_{ij} \cdot H(y_j, i)\right)) \cdot h_1^{(r_1+r_1')_j} \cdot r_{2j}'^{H(i)}, \prod_{j=1}^{n} g^{(r_1+r_1')_j}, \prod_{j=1}^{n} y_j \cdot h_2^{(r_1+r_1')_j}, \prod_{j=1}^{n} h_3^{(r_1+r_1')_j} \cdot r_{2j}' \right)$$

  - Finally, the node is asked to calculate a ZK proof in order to demonstrate that the Output Integrity Proof of one group is the re-encryption of the Input Integrity Proof of the same group.

  The node can proof that I_{IIi} and I_{Ioi} for a concrete group have the same content by:

    - Calculating

$$\frac{I_{Ioi}}{I_{IIi}} = \left(\prod_{j=1}^{n} h_1^{(r_1')_j} \cdot r_{2j}'^{H(i)}, \prod_{j=1}^{n} g^{(r_1')_j}, \prod_{j=1}^{n} h_2^{(r_1')_j}, \prod_{j=1}^{n} h_3^{(r_1')_j} \cdot r_{2j}' \right) = (u, v, w, z)$$

- Proving knowledge of the re-encryption factor $r_0' = \sum_{j=1}^{n} {r_1}'_j$ in *{v, w}* using the Schnorr Identification Protocol, being $g' = h_2$ and $h' = \prod_{j=1}^{n} {h_2}^{({r_1}')_j}$.

- Proving the knowledge of the re-encryption factor $r_0' = \sum_{j=1}^{n} {r_1}'_j$ in *{u, z}* using

$$g' = (h_3)/(h_1)$$

and

$$h_i' = \{s_i\} \cdot \frac{\prod_{j=1}^{n} {h_3}^{(r_1')_j}}{\prod_{j=1}^{n} {h_1}^{(r_1')_j}} = \left(\frac{\prod_{j=1}^{n} h_3}{\prod_{j=1}^{n} h_1}\right)^{r_0'}$$

where

$$s_i = \frac{\prod_{j=1}^{n} {r_{2j}}'}{\prod_{j=1}^{n} {r_{2j}}'^{H(i)}}$$

For the multiplication of the encrypted vote options a_i:

- The prover sends the factors $t = (\prod_{j=1}^{n} {r_{2j}}')^{r_0 - 1}$ and $k_i = (\prod_{j=1}^{n} {r_{2j}}'^{H(i)})^{r_0 - 1}$, for *i=1…k*, to the verifier.
- The verifier checks that $k_i = t^{H(i)}$, for each block (*i=1…n*).
- The prover and the verifier calculate

$$h_i'' = h_i' \cdot \frac{t}{k_i} = \left(\{s_i\} \cdot \frac{\prod_{j=1}^{n} h_3}{\prod_{j=1}^{n} h_1}\right)^{r_0'}$$

- Finally, the proof of knowledge of r_0' can be calculated using *g′* and h_i'' for each product of voting choices.

The steps to follow to calculate the proof and verify it are explained in Section 5.3.

- For the next node, new groups are redefined in such a way that a new group is composed of a vote from each of the old groups. Since an individual vote from a group could belong to any older group, votes can not be traced through the mix-nodes, preserving voter privacy. These steps are repeated for each node.

Note that we have considered the generalization the calculation of the *Integrity Proofs* as if they were *Intermediate Integrity Proofs* since this operation is performed over intermediate nodes too.

An example of this procedure is shown in the next figure:

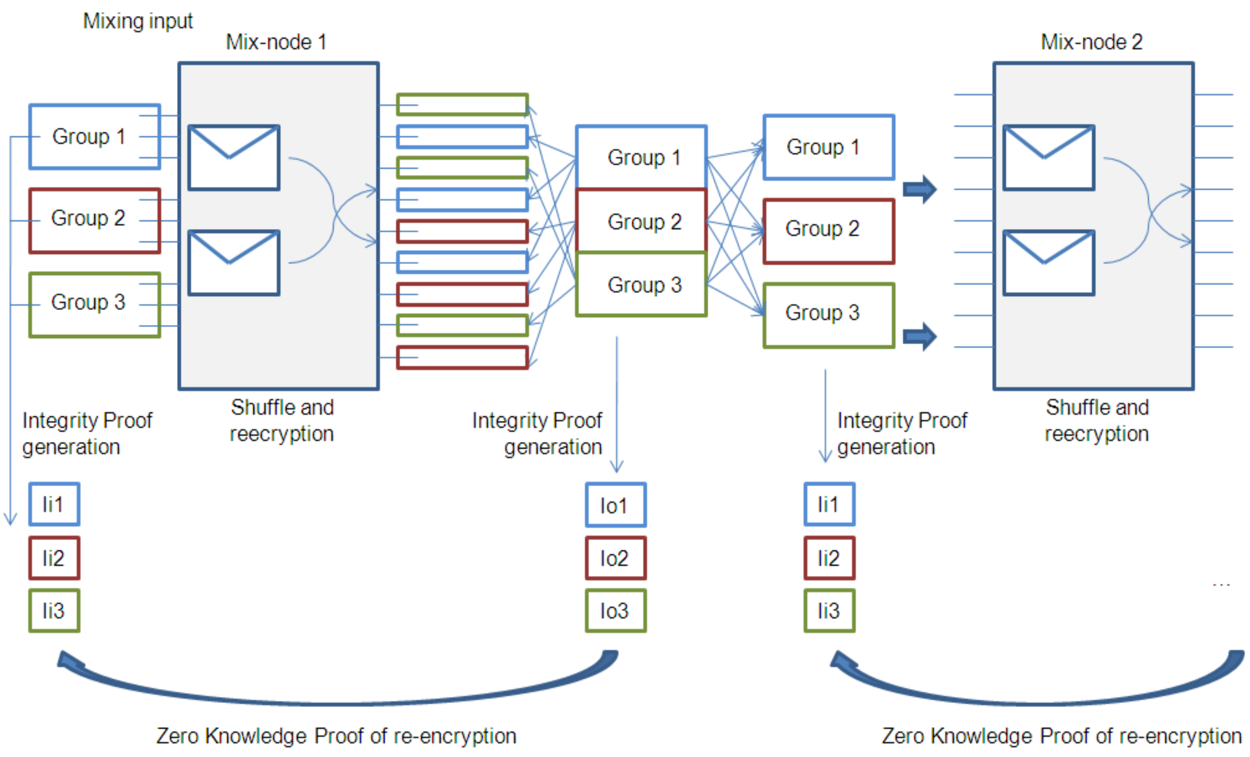


Fig. 6 - 3 Mixing verification procedure

Finally, we should explain that, in order to calculate the proof of re-encryption with the cryptosystem presented in the Proposal 6, an *Integrity Proof* must be calculated for each group of encrypted votes and for each encrypted voting option, as it is shown in the figure 6.4. Batch proofs can be used to optimize the process.

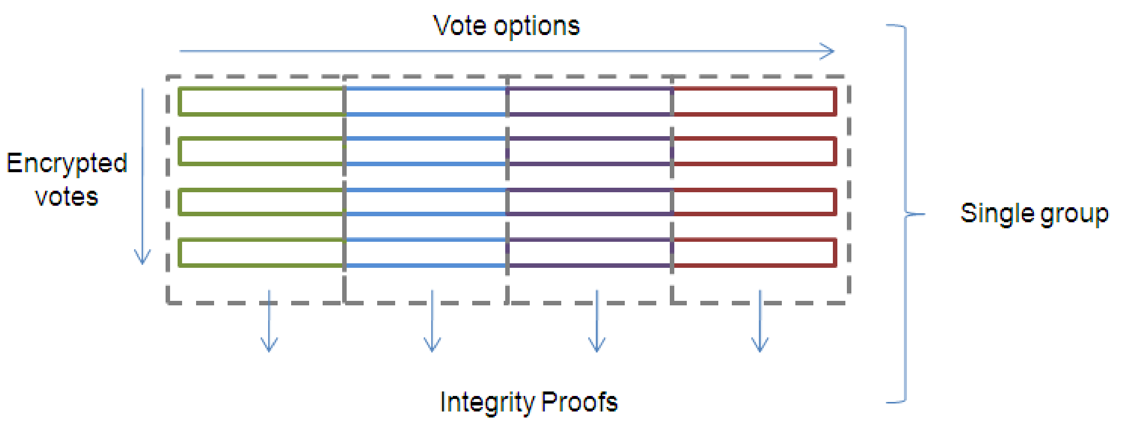


Fig. 6 - 4 *Integrity Proofs* calculation

In RGFC, instead of revealing individual traces between inputs and outputs (like in the RPC) the traces are revealed about groups. That way, the information about all the input messages can be asked (not just the half). At the same time provides a better anonymity than RPC, since the individual path of a vote is never disclosed.

Due to the Integrity Proof calculation, a cheater should have to change a pair of votes in such a way that the multiplication of the two modified votes is equal to the multiplication of the original ones (*"300 attack"*). Moreover, due to vote grouping, both modified votes should be in the same group when the check process is done. The cheater does not know which votes will belong to which group until the mixing has finished, due that this group-division is performed by a challenge from the Electoral Board.

In order to preserve voter privacy, the groups are configured in such a way that at the end of the mixing, although one could know to which group belongs in the last node a specific message, it could not be related to the group configuration in the first node. For this purpose, the group size is selected so that, being *k* the size of a group, $k = \sqrt[t]{n}$, where *t* is the number of nodes (at least two) and *n* is the total number of votes.

While in RPC just the half of the votes is verified at each node (having a probability of non-detection of 2^{-n} for *n* manipulated votes), using Random Group Full Checking we can divide the votes in more groups and audit the correct transformation of all of them. Then, the probability of non-detection is smaller.

The formula for obtaining the probability of non-detection for two manipulated votes is:

$$\frac{\binom{m-2}{n-2}}{\binom{m}{n}} \cdot g$$

where *m* is the total number of votes, *n* is the number of votes for each group, and *g* is the number of groups.

For example: in an election with 10.000 voters, where the votes are grouped in 100 groups in order to do the RGFC, the probability of non-detection of a manipulated couple is 0. 99%.

In an election with 1.000.000 voters, where the votes are grouped in 10.000 groups in order to do the RGFC, the probability of non-detection of a manipulated couple is 0. 01%.

6.4. Decryption and Counting Phase

Once all the re-encryption and shuffling steps in the mixing are verified, the votes are decrypted in the last node using the private keys from the Electoral Board. Each decryption step is verified using Schnorr ZK Proofs.

First, a partial decryption process is done before the votes are completely decrypted in order to verify the integrity of each message block using the hash attached to it.

Partial decryption

The received (re-encrypted) vote is

$$c' = (\{a_i' \ldots a_k'\},\ g^{r_1+r_1'}, y \cdot {h_2}^{r_1+r_1'}, r_2' \cdot {h_3}^{r_1+r_1'}) = (\mathrm{a}', \mathrm{b}', \mathrm{d}', \mathrm{e}'),$$

where

$a_i' = [m_i \cdot H(y,i)||H(m_i \cdot H(y,i))] \cdot {h_1}^{r_1+r_1'} \cdot {r_2}'^{H(i)}$, for each block.

- Calculate: ${h_3}^{r_1+r_1'} = (g^{r_1+r_1'})^{x_3}$ in order to obtain $r_2' = e' \cdot (g^{r_1+r_1'})^{-x_3}$.
- Calculate ${h_1}^{r_1+r_1'} = (g^{r_1+r_1'})^{x_1}$.
- Calculate ${r_2}'^{H(i)}$ for *i=1..n*.
- Then obtain

  $$a_{subi} = m_i \cdot H(y,i)||H(m_i \cdot H(y,i)) = \frac{a_i'}{{h_1}^{r1+r1'} \cdot {r_2}'^{H(i)}}$$

  for each vote option.

ZK Proof of partial decryption

The Schnorr ZK Proof can be done in order to proof the correct partial decryption of r_2' and a_{subi} :

- Calculate

  $$\frac{c}{[\{a_{subi} \cdot {r_2}'^{H(i)} \ldots a_{subn} \cdot {r_2}'^{H(n)}\}, 1,\ null, r_2']} = \left({h_1}^{r_1+r_1'}, g^{r_1+r_1'}, {h_3}^{r_1+r_1'}\right)$$

- Being ${h_1}' = {h_1}^{r_1+r_1'}$, ${h_3}' = {h_3}^{r_1+r_1'}$ and $g^{r_1+r_1'}$, the prover can demonstrate that he knows the private keys x_1, x_3, where ${h_1}' = g'^{x_1}$, ${h_3}' = g'^{x_3}$ using the Schnorr Identification Protocol:
  - The prover chooses a random z (the commitment), $1 \leq z \leq q-1$, and computes the witness $w = g'^{z}$.
  - The prover calculates a challenge (to make the protocol non-interactive) as *v=H(w/${h_1}'$/${h_3}'$/g'/h_1/h_2/h_3/g)*.
  - Finally, the prover calculates $s_1 = x_1 \cdot v + z$, $s_3 = x_3 \cdot v + z$, and sends *(w, s_1, s_3)* to the verifier.
- The verifier calculates ${h_1}' = {h_1}^{r_1+r_1'}$, ${h_3}' = {h_3}^{r_1+r_1'}$ and $g' = g^{r_1+r_1'}$ from the encrypted vote *c* and the parameters $[\{a_{subi} \cdot {r_2}'^{H(i)} \ldots a_{subn} \cdot {r_2}'^{H(n)}\}, 1,\ null, r_2']$ and calculates *v*.

- Then, he verifies that

$$g'^{s_1} = w \cdot {h_1}'^{v}$$
$$g'^{s_3} = w \cdot {h_3}'^{v}$$

Hash verification

Using the component

$$a_{subi} = m_i \cdot H(y,i)||H\big(m_i \cdot H(y,i)\big),$$

for each vote option:

- The verifier divides it in two pieces:

$$s = m_i \cdot H(y,i)$$

$$t = H\big(m_i \cdot H(y,i)\big)$$

- Then checks that *H(s)=t*.

If the verification process succeeds, the process can continue. Otherwise it is stopped and an error is reported.

Final decryption

- Calculate ${h_2}^{r_1+r_1'} = (g^{r_1+r_1'})^{x_2}$ in order to obtain $y = d' \cdot (g^{r_1+r_1'})^{-x_2}$
- For each encrypted vote option a_{subi} obtain m_i as

  $[m_i \cdot H(y,i)] \cdot H(y,i)^{-1}$.

ZK Proof of final decryption

The Schnorr ZK Proof can be done to proof the correct decryption of y and thus the final decryption:

- Calculate $c/[null, 1,\ y, null] = \big(g^{r_1+r_1'}, {h_2}^{r_1+r_1'}\big)$
- Being ${h_2}' = {h_2}^{r_1+r_1'}$ and $g' = g^{r_1+r_1'}$, the prover can demonstrate that he knows the private key x_2 , where ${h_2}' = g'^{x_2}$ using the Schnorr Identification Protocol:
  - The prover chooses another random *z'* (the commitment), $1 \leq z' \leq q-1$, and computes the witness $w' = g'^{z'}$.
  - The prover calculates a challenge (to make the protocol non-interactive) as *v'=H(w'/${h_2}'$/g'/h_1/h_2/h_3/g)*.

  - Finally, the prover calculates $s_2 = x_2 \cdot v' + z'$, and sends *(w',* $s_2)$ to the verifier.

- The verifier calculates ${h_2}' = {h_2}^{r_1 + r_1'}$ and $g^{r_1 + r_1'}$ from the encrypted vote *c* and the parameters y and calculates *v'*.

- Then, he verifies that

$$g'^{s_2} = w' \cdot {h_2}'^{v'}$$

Finally, once the decryption steps are verified for all the votes, the vote options m_i are counted and the results for each candidate are obtained.

7. Conclusions

The migration of elections from conventional environments to e-Voting requires the consideration of some security requirements, such as voter authentication, the preservation of voter privacy or the assurance of the legitimacy of the results.

There are standard cryptographic techniques that can be used in order to achieve these security requirements, such as the encryption and digital signature of the votes when they are submitted by the voters. In this case, we have seen that the use of digital envelopes is the best solution to encrypt the votes, since they take advantage of the simple key management methods of asymmetric encryption algorithms while they have the efficiency of the symmetric encryption systems. However, these standard cryptographic techniques are not enough to ensure voter privacy, since the signed and encrypted votes could be correlated with the decrypted ones and then the voter intent of vote could be disclosed. Thus, advanced cryptographic techniques must be used to preserve the voter privacy.

During the project, we analyzed some advanced cryptographic techniques that preserve the voter privacy by means of vote anonymization. From them, *Pollsterless* and two agencies models have some significant drawbacks concerning security and usability, so we discarded them in the beginning of the project. Therefore, we choose homomorphic tally and mixing systems as the best option to ensure the voter privacy during the election, according to our objectives. Evaluating both systems we finally decided that mixnets are the best system to use in our approach, since its combines high robustness with flexibility to work in different and complex election systems. Mixing systems, as well as any advanced cryptographic system, use verification methods to ensure their correct behaviour. These verification methods usually use advanced cryptographic techniques such as Zero Knowledge Proofs or other mathematical proofs based on the underlying cryptosystem, such as the *Integrity Proofs*. The requirement of using a cryptographic system supporting these advanced cryptographic techniques has been the main motivation of this project.

When these advanced cryptographic techniques are used, symmetric cryptosystems (or any technique based on them such as digital envelopes) can not be used, since these cryptosystems do not support the implementation of these advanced techniques. Therefore, asymmetric encryption algorithms are needed, preferably with homomorphic properties, to directly encrypt the information (i.e., digital envelopes are excluded since the information is encrypted with a symmetric key). Since asymmetric encryption methods are less efficient than symmetric ones from the point of view of encryption size (e.g., the encryption block size is usually larger in an asymmetric algorithm compared with a symmetric one of the same strength), and of decryption speed (due to the use of large keys), we focused the effort of this project on designing an asymmetric encryption method more efficient than the existing ones.

The first design requirement setup in the project was to improve the efficiency of the encryption/decryption process when the message exceeds the block size of the algorithm. However, during the process of designing this new asymmetric cryptosystem, some additional requirements arose, such as the need of supporting message re-encryption, homomorphic properties, be able to verify the integrity of the decrypted message or perform Zero Knowledge Proofs based on the designed cryptosystem. Furthermore, some attacks concerning the traceability of the messages through the mix-nodes had to be prevented. All these requirements have been considered from both the academic and the practical (real implementation) point of view.

The result is an efficient encryption/decryption cryptosystem that can be used in e-Voting environments or in other schemes that manages large messages and where the integrity of encrypted messages can be verified by means of advanced cryptographic techniques such as ZK Proofs.

Since Zero Knowledge Proofs are commonly used in e-Voting environments to achieve the desired security requirements, we have considered that it was important to suggest how could be performed these ZK proofs using the new encryption algorithm.

Finally, a case study is exposed to show how the designed cryptosystem could be used in a real e-Voting process verifying each election phase: voting, mixing, decrypting and counting.

Now, I would like to express some thoughts that came to me while I was doing this PFC:

Although the research field of cryptography in electronic voting environments has been active during many years and has produced lots of interesting protocols, there remains the question of if there is a system that accomplishes all the security requirements of an electoral process in a really efficient way. It is important to continue with this research and with the study of the newest protocols in order to achieve some day the fact of voting through the Internet being as common as buying a flight ticket, reaching a security level high enough to be able to fully trust in it .

Finally, I would like to say that the success of this PFC is not only the content you have read here, but also the learning on how to do research and investigate through the papers published in academic environments or that just fell into my hands.

8. References

[1] Chaum, D. (2001). *Surevote technical overview* (slides). http://www.vote.caltech.edu/wote01/pdfs/surevote.pdf

[2] Malkhi, D., Margo, O. and Pavlov, E. 2002. *E-Voting Without 'Cryptography'*. In: LNCS, 2357. pp. 1-15.

[3] Atsushi Fujioka, Tatsuaki Okamoto, and Kazui Ohta (1993). *A practical secret voting scheme for large scale elections.* In Advances in Cyptology - AUSCRYPT '92, volume 718 of Lecture Notes in Computer Science, pages 244--251, Berlin, 1993. Springer-Verlag.

[4] Benaloh, J. C. and Yung, M. 1986. *Distributing the power of a government to enhance the privacy of voters.* In Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing (Calgary, Alberta, Canada, August 11 - 13, 1986). PODC '86. ACM, New York, NY, 52-62

[5] R. Cramer, R. Gennaro and B. Schoenmakers. *A Secure and Optimally Efficient Multi-Authority Election Scheme*. In: Proceedings of EUROCRYPT '97, Konstanz, Germany, Springer Verlag LNCS, vol. 1233, pp. 103--118, May 1997.

[6] Peng, K; Aditya, R; Boyd, C, et al. *Multiplicative homomorphic E-voting.* Conference Information: 5th International Conference on Cryptology in India, Date: DEC 20-22, 2004 Chennai INDIA. Source: PROGRESS IN CRYPTOLOGY - INDOCRYPT 2004, PROCEEDINGS Volume: 3348 Pages: 61-72 Published: 2004

[7] Chaum, D. L. 1981. *Untraceable electronic mail, return addresses, and digital pseudonyms.* Commun. *ACM* 24, 2 (Feb. 1981), 84-90.

[8] K. Sako, J. Kilian. *Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth*, Advances in Cryptology -EUROCRYPT '95, Lecture Notes in Computer Science, Springer-Verlag, 1995.

[9] Ben Adida. *Advances in Cryptographic Voting Systems.* Thesis, 2006.

[10] Jakobsson, M., Juels, A., and Rivest, R. L. 2002. Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In *Proceedings of the 11th USENIX Security Symposium* (August 05 - 09, 2002). D. Boneh, Ed. USENIX Security Symposium. USENIX Association, Berkeley, CA, 339-353.

[11] D. Chaum. *Secret-Ballot Receipts and Transparent Integrity. Better and Less-costly Electronic Voting at Polling Places.* http://www.vreceipt.com/article.pdf

[12] Abe, M. 1999. *Mix-Networks on Permutation Networks.* Lecture Notes In Computer Science, vol. 1716. Springer-Verlag, London, 258-273.

[13] Markus, J. and Ari, J. 1999 *Millimix: Mixing in Small Batches*. Technical Report. UMI Order Number: 99-33., Certer for Discrete Mathematics & Theoretical Computer Science.

[14] Neff, C. A. 2001. *A verifiable secret shuffle and its application to e-voting*. In Proceedings of the 8th ACM Conference on Computer and Communications Security (Philadelphia, PA, USA, November 05 - 08, 2001). P. Samarati, Ed. CCS '01. ACM, New York, NY, 116-125.

[15] Golle, P., Zhong, S., Boneh, D., Jakobsson, M., and Juels, A. 2002. *Optimistic Mixing for Exit-Polls.* In Proceedings of the 8th international Conference on the theory and Application of Cryptology and information Security: Advances in Cryptology (December 01 - 05, 2002). Y. Zheng, Ed. Lecture Notes In Computer Science, vol. 2501. Springer-Verlag, London, 451-465.

[16] Wikstrom Douglas. *Five practical attacks for optmistic mixing for exit-polls.* In: Proceedings of SAC 2003, 2003. p. 160-75.

[17] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press. ISBN: 0-8493-8523-7. October 1996, 816 pages

[18] www.keylength.com

[19] Adida, B. 2008. *Helios: web-based open-audit voting.* In Proceedings of the 17th Conference on Security Symposium (San Jose, CA, July 28 - August 01, 2008). USENIX Association, Berkeley, CA, 335-348.

[20] http://www.bouncycastle.org/java.html

[21] Schneier, Bruce. *Applied Cryptography*, John Wiley & Sons, 1994. ISBN 0-471-59756-2

[22] Sandler, D., Derr, K., and Wallach, D. S. 2008. *VoteBox: a tamper-evident, verifiable electronic voting system.* In Proceedings of the 17th Conference on Security Symposium (San Jose, CA, July 28 - August 01, 2008). USENIX Association, Berkeley, CA, 349-364.

[23] B. Pfitzmann. *Breaking efficient anonymous channel.* In A. D. Santis, editor, Advances in Cryptology (Eurocrypt '94), volume 950 of LNCS, pages 332--340, Perugia, Italy, 9-12 May 1994. Springer-Verlag.

[24] Park, C., Itoh, K., and Kurosawa, K. 1994. *Efficient anonymous channel and all/nothing election scheme.* In Workshop on the theory and Application of Cryptographic Techniques on Advances in Cryptology (Lofthus, Norway). T. Helleseth, Ed. Springer-Verlag New York, Secaucus, NJ, 248-259.

[25] Hwang, M. S., Chang, C. C., and Hwang, K. F. 2002. *An ElGamal-Like Cryptosystem for Enciphering Large Messages.* IEEE Trans. on Knowl. and Data Eng. 14, 2 (Mar. 2002), 445-446.

[26] Wang, M., Yen, S., Wu, C., and Lin, C. 2006. *Cryptanalysis on an Elgamal-like cryptosystem for encrypting large messages.* In Proceedings of the 6th WSEAS

international Conference on Applied informatics and Communications (Elounda, Greece, August 18 - 20, 2006).

[27] Zhong, S. 2006. *An Efficient and Secure Cryptosystem for Encrypting Long Messages.* Fundam. Inf. 71, 4 (Mar. 2006), 493-497.

[28] Tata Elxsi. *Elliptic Curve Cryptography – An Implementation Tutorial*, India, January 5, 2007.

[29] http://en.wikipedia.org/wiki/Elliptic_curve_cryptography

[30] D. Hankerson, A. Menezes, and S.A. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag, 2004.

[31] N. Koblitz, *Elliptic curve cryptosystems*, in *Mathematics of Computation* 48, 1987, pp. 203–209

[32] http://www.secg.org/download/aid-386/sec2_final.pdf

[33] A. Menezes, T. Okamoto, and S.A. Vanstone, *Reducing elliptic curve logarithms to logarithms in a finite field,* IEEE Transactions on Information Theory, Volume 39, 1993.

[34] http://www.secg.org/

[35] http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf

[36] Josep María Miret Biosca. *Criptografía con Curvas Elípticas.* Universidad de Lleida (España). Enero de 2005. Primer Congreso Conjunto de Matemáticas RSME-SCM-SEIO-SEMA mat.es 2005.

[37] www.secg.org/collateral/**sec1**_final.pdf

[38] http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

[39] P Paillier. *Public key cryptosystems based on composite degree residuosity classes*. In Proc.of Eurocrypt '99, volume 1592 of LNCS, pages 223 -238. IACR,Springer-Verlag, 1999.

[40] S. Galbraith, *Elliptic curve paillier schemes*, Journal of Cryptology , vol. 15, no. 2, pp. 129-138, 2002.

[41] D. Galindo, S. Martin, P. Morillo and J.L. Villar. *An efficient semantically secure elliptic curve cryptosystem based on KMOV scheme.* International Workshop on Coding and Cryptography WCC 2003. Versailles, France.

[42] C. Munuera, J. Tena : *Codificacion de la Informacion*, U. de Valladolid, 1997.

[43] O. Ugus, A. Hessler and D. Westhoff, *Performance of Additive Homomorphic EC-ElGamal Encryption for TinyPEDS*, RWTHAachen, 2007.

[44] Sebastià Martín Molleví. *Curvas elípticas módulo N y aplicaciones criptográficas*. Tesis Doctoral. Universidad Politécnica de Cataluña (España), 1998.

[45] R. Schoof: *Counting Points on Elliptic Curves over Finite Fields*. J. Theor. Nombres Bordeaux 7:219-254, 1995. Available at http://www.mat.uniroma2.it/~schoof/ctg.pdf

[46] Jean-Jacques Quisquater, Louis C. Guillou, Thomas A. Berson. *How to Explain Zero-Knowledge Protocols to Your Children.* Advances in Cryptology - CRYPTO '89: Proceedings, v.435, p.628-631, 1990

[47] Golle, P. 2005. *Dealing Cards in Poker Games*. In Proceedings of the international Conference on information Technology: Coding and Computing (Itcc'05) - Volume I - Volume 01 (April 04 - 06, 2005). ITCC. IEEE Computer Society, Washington, DC, 506-511.

[48] Francesc Sebé, Josep M. Miret, Jordi Pujolàs, Jordi Puiggalí. *Simple and Efficient Hash-based Verifiable Mixing for Remote Electronic Voting*. Computer Communications (17 November 2009)

[49] M. Jakobsson*. A practical mix*. In K. Nyberg, editor, EUROCRYPT '98, pages 448-461. Springer-Verlag, 1998. LNCS No. 1403.

[50] Fiat, A. and Shamir, A. 1987. *How to prove yourself: practical solutions to identification and signature problems.* In Proceedings on Advances in cryptology---CRYPTO '86 (Santa Barbara, California, United States). A. M. Odlyzko, Ed. Springer-Verlag, London, 186-194.

[51] Chaum, D. and Pedersen, T. P. 1993. *Wallet Databases with Observers*. In Proceedings of the 12th Annual international Cryptology Conference on Advances in Cryptology (August 16 - 20, 1992). E. F. Brickell, Ed. Lecture Notes In Computer Science, vol. 740. Springer-Verlag, London, 89-105.

[52] Storer, Timothy W. *Practical Pollsterless Remote Electronic Voting.* Thesis, 2007.