New approaches for electronic voting paradigms

Víctor Mateu Meseguer

TESI DOCTORAL UdL · 2015

New Approaches for Electronic Voting Paradigms

Víctor Mateu Meseguer

Directors: Josep M. Miret i Francesc Sebé

Programa de Doctorat en Enginyeria i Tecnologies de la Informació

Universitat de Lleida

Resum

La democràcia és el sistema de govern més utilitzat al món. No obstant, en un món cada vegada més globalitzat, la idea de mobilitzar la gent per votar en un col.legi electoral gestionat per persones resulta antiquada tot i ser la implementació més comú en l'actualitat. Millorar aquesta situació mitjançant l'ús de les tecnologies de la informació sembla una evolució òbvia i molt demanada però, malgrat l'existència d'algunes implementacions en entorns reals, encara no ha estat utilitzada excepte en comptades ocasions.

Obrir la porta d'unes eleccions a les tecnologies de la informació implica l'obertura dels protocols de votació a un nou conjunt d'atacs contra aquests. Tenint en compte els requisits d'una elecció: privacitat del votant i integritat de l'elecció, les solucions actuals passen per implementar l'elecció seguint un dels tres paradigmes de vot segurs: barreja de vots, recompte homomòrfic o signatura cega.

En aquesta tesi, es proposen nous protocols per als diferents paradigmes.

La primera proposta consisteix en un sistema de vot que, basant-se en una informació redundant enviada pel votant, és capaç de realitzar una barreja de vots amb cost negligible incrementant lleugerament el cost del recompte.

Per al paradigma de recompte homomòrfic, es proposa una prova de validesa del vot basada en les proves utilitzades per demostrar la correctesa en sistemes amb barreja de vots. Aquesta solució permet utilitzar les millores realitzades sobre el paradigma de barreja de vots per al seu ús en el paradigma de recompte homomòrfic.

Finalment, es plantegen dues solucions per a eleccions del paradigma de signatura cega. La primera utilitza credencials generades amb signatura cega per permetre als votants vàlids enviar el seu vot sense que es conegui la seva identitat. La segona resol el problema del vot doble en aquest paradigma mitjançant una construcció que utilitza un sistema de moneda electrònica off-line.

Resumen

La democracia es el sistema de gobierno más usado en el mundo. No obstante, en un mundo cada vez más globalizado, la idea de movilizar a la gente para votar en un colegio electoral gestionado por personas resulta anticuada a pesar de ser la implementación más común en la actualidad. Mejorar esta situación mediante el uso de las tecnologías de la información parece una evolución obvia y muy solicitada pero, a pesar de unas pocas adaptaciones, aún no ha sido usada salvo en escasas ocasiones.

Abrir la puerta de unas elecciones a las tecnologías de la información lleva implícita la apertura de los protocolos de voto a un nuevo conjunto de ataques contra estos. Teniendo en cuenta los requisitos de una elección: privacidad del votante e integridad de la elección, las soluciones actuales pasan por implementar la elección siguiendo uno de los tres paradigmas de voto seguros: mezcla de votos, recuento homomórfico o firma ciega.

En esta tesis, se proponen nuevos protocolos para los distintos paradigmas.

La primera propuesta consiste en un sistema de voto bajo el paradigma de mezcla de votos que, basándose en una información redundante enviada por el votante, es capaz de realizar una mezcla de votos con un coste negligible incrementando ligeramente el coste del recuento.

Para el paradigma de recuento homomórfico, se propone una prueba para verificar que el voto es válido basada en las pruebas de correctitud en sistemas con mezcla de votos. Esta solución permite usar las mejoras realizadas en el paradigma de mezcla de votos para su uso en el paradigma de recuento homomórfico.

Finalmente, se proponen dos nuevos protocolos del paradigma de firma ciega. El primero utiliza credenciales generadas con firma ciega para permitir a votantes válidos enviar su voto sin que se conozca su identidad. El segundo resuelve el problema del voto doble en el paradigma de firma ciega mediante una construcción que utiliza un sistema de moneda electrónica off-line.

Abstract

Democracy is the most established government system in the world. However, in an increasingly globalized world, the idea of requiring people to move in order to cast their vote in the polling station seems outdated, even though it is, nowadays, the most common implementation. An obvious and widely demanded evolution is to improve the election framework by enabling the use of information technologies. Nevertheless, this solution has been implemented few times in real environment elections and the global success of these solutions have been called into question.

The use of information technologies in voting protocols improves the quality of the election but, at the same time, it also opens up the voting protocols to new threats. Keeping this attacks in mind and given the election requirements: voter's privacy and election's integrity, the solutions proposed up to date are to implement one of the three secure voting paradigms: mix-type based, homomorphic tally, and blind signature.

In this thesis, we present new protocols for the different paradigms.

Our first proposal, based on the mix-type paradigm, consists in a voting protocol which is able to perform the ballot mix with negligible cost but slightly increasing the tally cost. The proposed protocol makes use of a proper vote generation based on sending secret redundant information with the ballot when it is cast.

For the homomorphic tally paradigm, we propose a zero knowledge proof of correctness of the ballot based on the proofs used to demonstrate the correctness of a shuffle in the mix-type paradigm. This protocol makes possible to use the improvements on the shuffle correctness proofs in the homomorphic tally paradigm.

Finally, two different protocols are also proposed for the blind signature paradigm. The first one uses credentials generated by means of a blind signature which allow eligible voters to cast their vote without leaking information about their identity. The second one is focused on solving the double voting problem in this paradigm. The protocol proposed uses off-line e-coin systems to provide anonymity disclosure in case of double voting.

Acknowledgements

First of all, I would like to thank my PhD advisors for giving me the opportunity to learn and work with them. It has been a unique experience and they deserve all the credit for that. I also want to thank Santi, Rosana, Núria, Javi, M. Àngels and Ricard for all the hours of discussion in the office. You have seen the evolution of this thesis and you have also participated in each one of the papers, it was nice to share our points of view and try to solve the problems of the world from the office. I want to extend my gratitude to all the members of the Departament de matemàtica, particularly to Magda for all her advices.

Another ones who deserve special mention are my parents and my brother. They have been there all the time, supporting and caring about everything.

I could not forgive myself if I do not include Sonia. You have done everything for me and I really appreciate it.

I would like to thank Enric for allways being there, sharing the best and the worst of us, I know for sure that great part of my motivation was thanks to him.

Finally, I would also like to show my gratitude to Prof. Jeroen van de Graaf for opening my mind to a new research topic and to Prof. Jörn Müller-Quade for allowing me to have a research stay with him and his team. I could not forget to mention Carmen Kempka, we shared lots of ideas and I have learnt alot from her and from all the research group during my stay in the Karlsruhe Institute of Technology.

This thesis has received a grant for its linguistic revision from the Language Institute of the University of Lleida (2015 call)

Contents

Chapter 1

Introduction

Human beings have always been arranged according to some social structure. Nowadays, most civilizations in the world are democratic. Democracy was born in Athens in the 5th century. The most distinctive aspect of that democracy was the fact that decisions were taken through a deliberative assembly in which only Athenian males could participate. This restriction caused the assembly to include only 25 percent of the city population. The low population and the small city size allowed this kind of social governance.

When the population increased, the capacity to control the city diminished significantly and the government structure had to be adapted in order to handle the new situation.

Nowadays, the most extended solution is a democracy-based approach in which a government composed of a few citizens is chosen through an electoral process. These representatives take the government decisions.

An electoral process consists of three phases. The first one is devoted to the composition of a list containing all the people with the right to participate in the election. These people are the eligible voters, and the list generated is called the electoral roll. Composing this list is a hard task. It takes considerable time and it is hard to avoid mistakes during its generation. The use of information technologies has highly improved this process. Nowadays, the electoral roll can be generated automatically using the administration software.

The second phase of an electoral process comprises all the tasks related to gathering the voters' choices, usually referred to as their ballots. This includes the planning of a proper distribution of ballot boxes and their closure at the moment indicated. This step requires substantial resources. A great deal people are required to guard the ballot boxes and guarantee that they have not been improperly manipulated. Moreover, the infrastructure required is considerable for a city or a country election. The economic cost of this phase is rather elevated.

The last step focuses on tallying the election results. The ballot boxes

are opened and each ballot has to be tallied. The time required for this step depends on the amount of ballots and the amount of people counting. If the amount of people involved increases, the result of the election can be gathered faster. However, when the people involved increases, guaranteeing the correctness becomes harder.

The cost of an election is large in terms of people involved and places and material required. Moreover, it also requires a large number of trusted individuals to ensure the correctness of the election. Even with these drawbacks, the majority of countries use this method to choose the representatives for their governments.

1.1 Remote electronic voting

Electronic voting systems substitute some components of a traditional election with an electronic process. An electronic voting system in which voters can cast their votes remotely through telecommunication networks will be referred to as a *remote voting system*.

In an election, it is essential that voters can prove their identity. In traditional elections carried out in a polling station, voters usually identify themselves by showing an ID card. In remote voting systems, the most appropriate identification method is the use of *digital certificates*. A digital certificate is a digital set of data containing a public key and some personal data which identifies the certificate holder. This data is signed by a trusted authority. A person authenticates herself by transmitting the digital certificate and demonstrating that she is in possession of the private key related to the public key in the certificate. This can be done, for example, by computing a digital signature on some random message.

Traditional paper-based elections offer the opportunity to vote easily and provide a feeling of transparency. The security of these elections is based on the supervision of the process carried out by several inspectors. In contrast, with remote voting, the participants can only see the interface of the voting software and they are not therefore aware of internal details. This situation can generate a lack of trust. A remote voting system must guarantee and be able to prove the correctness of an election result to any suspicious observer.

Both in the traditional and in the remote cases, an election involves a set of actors with the following roles:

- **Voters:** Each person whose name appears on the *electoral roll* is an eligible voter. These are the only people allowed to cast a ballot. A voter creates her ballot and deposits (in traditional voting) or transmits (in remote voting) it to the ballot collection authority.

- **Electoral roll authority:** This authority creates the electoral roll and makes it publicly accessible so that the voters can check it and

ask for modifications if needed. After that, the electoral roll is made available for the election.

- **Ballot collection authority:** This authority receives voters' ballots. It first asks the voter to identify herself and checks whether she is listed on the electoral roll (authentication). After that, it verifies that the authenticated voter has not cast a ballot before (unicity). If all these checks are satisfied, the ballot is stored. In a traditional election, ballots are put in a ballot box. Remote voting systems make use of a publicly accessible *bulletin board* to publish the ballots received in an encrypted form. This is required for *verifiability* purposes.

- **Vote tallying authorities:** At the end of the election, these authorities open the ballots, tally the votes and publish the results. It is frequent to refer to the set of authorities as Key Storage Trusted Party (KSTP).

Some remote voting proposals require additional authorities. For instance, in the blind signature paradigm, authentication is not carried out by the ballot collection authority but by an *authentication authority*.

There are other authorities which are relevant for a voting system, even though they are not directly related to it. For example, in remote voting, voters may authenticate themselves by means of digital certificates issued by an external *certificate authority*. A similar entity is also needed in traditional elections, *i.e.* the ID card issuer.

These parties are involved in the different phases of an election. The phases in a remote voting election are very similar to those of a traditional election. Different remote voting systems implement them in many ways, but the main actions per phase remain very similar. The phases occur sequentially: the next phase does not start until the previous one has ended. They are described below.

1. **Setup:** During this phase, the parameters needed to run the election are generated and verified. These parameters include:

   - The electoral roll, which is published on a bulletin board so that anyone can check it and ask the electoral roll authority to correct possible mistakes.
   - The election keys, generated by the tallying authorities. These keys will be used for vote encryption so that votes can be cast privately.

   Moreover, the voters should check their certificates in order to ensure their validity, checking, for instance, that they are not out of date.

2. **Vote casting:** In this phase, the voters compose and cast their ballots. A voter first generates a message representing her vote and encrypts it under the election public key. After that, she signs her ballot and sends it together with its digital signature and her digital certificate to the ballot collection authority. Upon reception, this authority checks that:

   - The digital certificate is valid and belongs to a voter on the electoral roll.
   - The ballot is signed under the key on the digital certificate.
   - The voter has not cast a ballot before.

   If all these checks are passed, the ballot collection authority publishes the ballot, its signature and the voter's certificate on a publicly accessible bulletin board so that any external entity can perform the same checks.

3. **Tallying:** When the vote casting phase has concluded, the vote tallying authorities take the ballots from the bulletin board and decrypt them. After that, the results of the election can be published. The decryption of each ballot has to be performed verifiably in the sense that any external entity can check that each cleartext vote really comes from the decryption of a ballot.

Most remote voting schemes implement these three phases. Nevertheless, some proposals include some additional phase. For example, one of our proposals includes an additional *vote credential request* phase.

1.2 Security requirements

An election is a process designed to gather the opinion of voters. A voting system must ensure that:

- Only people on the electoral roll can vote (authentication).
- Nobody knows the individual vote of any voter (privacy).
- Nobody can cast more than one ballot (unicity).

The purpose of these requirements is to encourage people to vote. Remote voting provides all the advantages of information technologies to an electoral process. Making an election accessible through the Internet allows voters to cast their ballots without the need to go to the polling place. Nevertheless, this does have drawbacks. A plethora of new attacks (denial of service, voter impersonation, ballot modification,...) is now possible. For

this reason, the original security requirements have to be more accurately defined and expanded. The use of frameworks [JCPACRV12] and game-based security definitions [BCG+15] to evaluate remote voting schemes is increasingly demanded as the amount of proposals increases. These frameworks and security definitions are based on the well-known security requirements of an election which are detailed below.

1.2.1 Integrity

In remote voting, *integrity* refers to *correctness* (only voters on the electoral roll can vote one time at most) and *fairness* (the parties involved cannot successfully misbehave). In order to provide these properties a voting system must ensure the following:

- **Voter authentication:** Each vote received and tallied has been cast by a voter who appears on the electoral roll.
- **Unicity:** There is at most one tallied vote per voter.
- **Fairness:** It is not possible to add any fraudulent vote to the tally. Moreover, the votes received cannot be removed or modified during the election process.

An electronic voting system must provide integrity so that the result of the election cannot be altered in any way. This means that all the votes have been cast as they were intended to be by eligible voters and, after that, the votes cannot be modified during their processing.

In a traditional election, integrity is provided by means of trusted supervisors. In remote voting systems, the integrity requirement is highly related to verifiability. The system has to provide some evidence able to convince an external verifier that no manipulation has been carried out.

Verifiability

Verifiability is defined as *"the quality or state of being capable of being verified, confirmed, or substantiated"*.

An election is verifiable when any entity can check that the election result really comes from the votes cast by the participants. The election result has to be proven to be correct even when there is the possibility of misbehaviour by some party. If a misbehaving act happens, it must be detected.

Traditional elections provide verifiability by means of audit authorities. However, this solution does not allow anyone but the aforementioned authorities to verify anything. The integrity of the election depends on trusting each of these audit authorities. However, it is not verifiable since some of them could misbehave without being detected.

In contrast, remote voting systems can provide public verifiability in the sense that a misbehaving act will be detected. Current remote electronic voting schemes are able to provide what it is called *end-to-end verifiability*. An election is end-to-end verifiable when all the phases of the election can be verified.

1.2.2 Privacy

Privacy is defined as *"the ability of an individual or group to keep information secret"*. When something is private to a person, it usually means that something is inherently special or sensitive to her. Privacy is a key aspect in elections. It allows the voters to freely express themselves. This is specially important in elections with confronting options like referendums in which a voter's opinion leakage could be harmful to her.

Traditional elections provide privacy by putting the vote into a standard paper envelope. This vote plus its envelope is the ballot of a voter. When the voting phase ends, the ballots stored in ballot boxes are mixed in order to break any link with the voters who cast them.

In a remote voting system, privacy is provided by means of cryptographic techniques. There are two properties to take into account:

- **Confidentiality:** The voting process must ensure that the identity of a voter cannot be linked to her vote. When a voter encrypts her vote, the resulting ballot leaks no information about the content. However, an election requires the ballot to be signed in order to authenticate its caster. In this way, when the ballot is decrypted the vote could be related to the voter who cast it. Hence, in remote voting systems the link between a ballot and its signature has to be broken somehow prior to ballot decryption.

- **Coercion resistance:** Voters are not able to prove how they voted, despite being able to provide evidence that they have cast a vote. The fact that there is evidence of their votes would make them easier to be coerced. Some coercion-resistance techniques provide tools allowing voters to cheat the coercers. Coercers are tricked into thinking that the voter has behaved under coercion.

1.2.3 Robustness

Robustness is defined as *"the ability of a system to resist changes without adapting its initial stable configuration"*.

Technically, it cannot be considered a security requirement. Nevertheless, even without being a requirement, it is a property of which to be aware.

Some elections have had issues regarding irregular envelopes which failed to provide confidentiality to voters.

This property includes the system's strength against certain kinds of attack. The system should be able to face situations in which some voters or authorities are misbehaving in order to disrupt the process. A remote voting system must provide integrity and privacy in order to be secure. However, these requirements are not enough if the election servers become out of order due to a Denial of Service (DoS) attack.

Along the same lines, a remote voting system should provide an elevated Quality of Service (QoS). That is, the software can be executed easily on several types of device providing an enjoyable experience for the user. Also, the servers should respond to queries within a short period of time.

1.3 Remote voting paradigms

Nowadays, remote voting schemes provide secure platforms to vote through the Internet. In order to design a secure e-voting scheme, the distribution of tasks as well as the configuration of the roles have to be accurate. There are plenty of critical data distributed among different entities which are sent and linked to public information. Over the years, three main paradigms have emerged as suitable solutions. These paradigms are:

- **Mix-type voting paradigm:** This is the most similar to traditional elections. During the setup phase, the tallying authorities generate the private key of the election and publish the corresponding public key. This public key is used by the voters to encrypt their votes. After that, the vote casting phase starts, and the voters send their ballots (encrypted votes) digitally signed to the ballot collection authority.

  When this phase has ended and before proceeding with the tallying phase, there is a *mixing phase* in which a party of mixers takes the ballots from the bulletin board and re-encrypts and shuffles them. In this way, a new set of ciphertexts is generated by the mixers. These ciphertexts are encrypting the same votes as the ballots cast, but they cannot be related to the signatures received. Finally, during the tallying phase, the tallying authorities decrypt the mixed ballots and publish the election result.

- **Homomorphic tallying paradigm:** This paradigm implements the election phases (setup, vote casting and tallying) without adding any additional one. During the setup phase, an election public key is generated. The public key cryptosystem employed is required to have a homomorphic property.

  When the vote casting phase starts, each voter generates her vote and encrypts it under the homomorphic cryptosystem. After that, she signs it and sends the ballot, her signature and her digital certificate

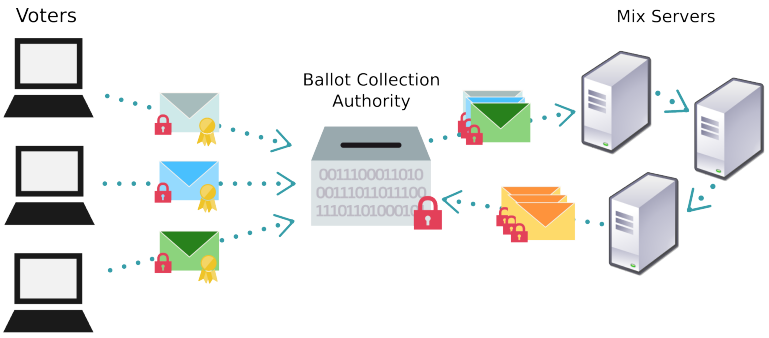

Figure 1.1: Mix-type paradigm

to the ballot storing authority. If everything is correct, the authority stores the ballot and publishes it together with the signature on the bulletin board.

Once all the ballots have been received or the vote casting phase has ended, the tallying authorities homomorphically aggregate (using the cryptosystem homomorphic operation) all the ballots and proceed with the decryption of the resulting ciphertext. An appropriate coding of election options allows the decryption of this aggregated ciphertext to result in a new message from which it is possible to gather the election results. After that, the tallying authorities publish the results. Notice that, as a result of the homomorphic aggregation, a new ciphertext which encapsulates the information of all the ballots is generated. In this way, when decrypted no one is able to distinguish which value came from each ballot and the link between the identity of a voter and her vote is broken.

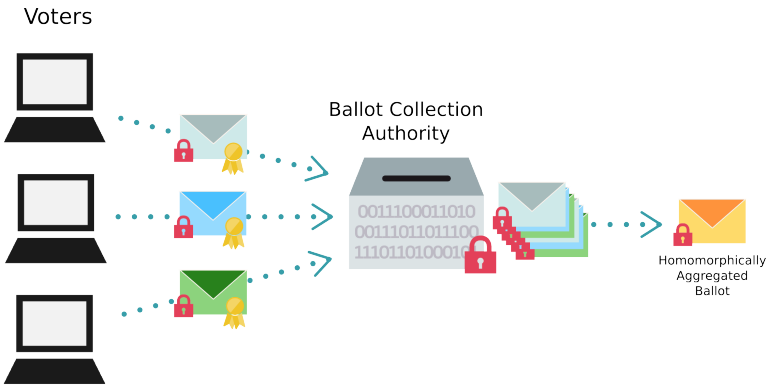

Figure 1.2: Homomorphic tallying paradigm

- **Blind signature paradigm:** With this paradigm, the authentication and ballot collection tasks are performed by separate entities. Voter

authentication is carried out by an *authentication authority*. This party must be trusted by all the participants. Implementing it as a distributed authority provides a better trust in it.

During the setup phase, the following actions are carried out:

- The electoral roll is generated and published.
- A private / public key pair for the authentication authority is generated. These keys will be used to blindly sign the ballots of authenticated voters.
- A private / public election key pair for the tallying authority is generated. This key pair will be used for vote encryption.

During the vote casting phase a voter composes her vote and encrypts it under the election public key. After that, she authenticates against the authentication authority who checks that the voter appears on the electoral roll and has not voted before. After that, the voter and the authentication authority run a *blind signature protocol*. As a result, the voter obtains a signature of her ballot under the authentication authority public key, while the authentication authority obtains no information about the voter's ballot. The voter can check that the resulting signature is correct by verifying it under the authentication authority public key.

After that, the voter sends her ballot together with the signature from the authentication server to the ballot collection authority through an anonymous channel. The use of an anonymous channel is required to avoid an eventual tracing of the vote provenance. For instance, the voter could be identified from her device IP address. An anonymous channel makes the ballots received untraceable. The vote collection authority will only store and publish ballots that carry a valid signature computed by the authentication server.

When the vote casting phase ends, the tallying authorities decrypt each ballot and publish the election results.

In the previous overview, we have not mentioned that the proposals implementing the aforementioned remote voting paradigms usually require the use of zero-knowledge proofs that ensure the correctness of the process and permit its verifiability. The computational cost of these proofs is usually the most important aspect to take into account when designing new proposals.

1.4 Contributions

In this thesis, several contributions to remote voting are presented. They are summarized below taking into account the paradigm to which each of them belong.

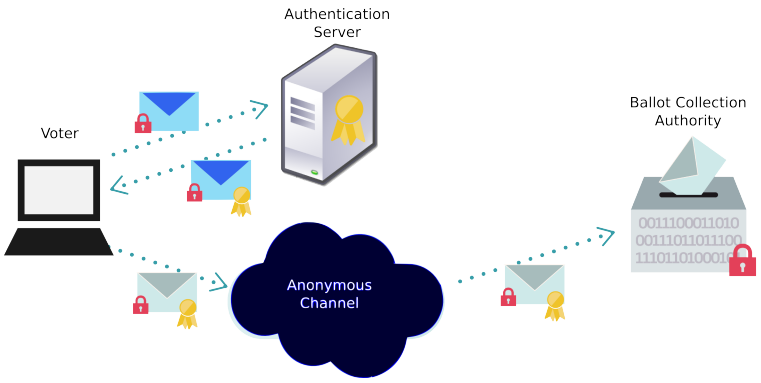

Figure 1.3: Blind signature paradigm

Contribution to the mix-type paradigm

This paradigm accommodates very flexible remote voting systems. Its drawback is located in the zero-knowledge proof needed to ensure the correctness of the mixing process. The proofs proposed in the literature usually have high computational and conceptual complexities.

Our proposal *"Verifiable encrypted redundancy for mix-type remote electronic voting"* [MMS11] provides an efficient method to prove the correctness of a mixing operation. The proposal follows a "proof of product with redundancy" [GZB+02] approach. Votes are composed using a redundancy system that permits proof of the correct composition and plaintext awareness of votes at vote reception. Moreover, it also makes it possible to check the validity of the mixing operation while preventing some privacy or denial of service attacks that were possible in previous proposals in the literature.

Contribution to the homomorphic tallying paradigm

Vector-based homomorphic tallying remote voting schemes provide an efficient vote tallying procedure, but they require voters to prove in zero-knowledge that the ballots they cast have been properly generated. This is usually achieved by means of the so-called zero-knowledge range proofs which should be verified by the polling station before tallying. In our proposal: *A hybrid approach to vector-based homomorphic tallying remote voting* [MMS15], we present an end-to-end verifiable hybrid proposal in which ballots are proven to be correct by making use of a zero-knowledge proof of mixing but still using homomorphic tallying to collect the election results. Our proposal offers all the advantages of the homomorphic tallying paradigm while avoiding the elevated computational cost of range proofs. As a result, ballot verification performance is improved in comparison with the equivalent homomorphic systems. The proposed voting scheme is suitable for multi-candidate elections as well as for elections in which the voters have

different weights.

Contribution to the blind signature-based paradigm

Blind signature-based electronic voting is the simplest paradigm to implement remote voting platforms due to the fact that it does not employ complicated zero-knowledge proofs. Unfortunately, the existence of a trusted entity (the "Authentication Server") that, in case of corruption, would be able to cast indistinguishable fake votes, reduces the acceptance of the paradigm in non-fully trusted environments. Trust in the system can be increased by splitting this entity into of a set of parties that are unlikely to collaborate in a dishonest manner. Nevertheless, this technique increases the risk of failure of some of them, causing a service interruption during the voting period. Better fault tolerance is provided by proposals which make it possible to anticipate the interaction with the distributed authentication server before the voting period begins, so that, in case of failure, there is a broad time margin for system restoration.

Previous proposals following this approach have been proven to be cryptographically weak or just provide individual verifiability. In *Blind Certificates for Secure Electronic Voting* [MSV13], we present a system that employs blind certificates. Unlike previous proposals, it provides universal verifiability and permits us to detect double voting without putting voters' privacy at risk.

Double voting detection for blind-signature remote voting was introduced by Mu and Varadharajan. They proposed a remote voting paradigm in which participants receive a blindly signed voting credential that permits them to cast a vote anonymously. If some participant tries to cheat by submitting more than one vote, her anonymity will be lifted. In recent years, several proposals following this paradigm, including Mu and Varadharajan's, have been shown to be cryptographically weak. In *Constructing credential-based E-voting systems from offline E-coin protocols* [MSV14], we first show that a recent proposal by Baseri *et al.* is also weak. After that, we give a general construction that, employing an offline e-coin protocol as a building block, provides a secure anonymous voting system following the aforementioned paradigm.

1.5 Structure

The thesis is divided into six chapters. The first one is devoted to the introduction of remote electronic voting. There is a brief historical explanation of democracy which follows with a description of the roles and phases of an electronic election in Section 1.1. Section 1.2 is devoted to defining the security requirements of an electronic election. The main paradigms fulfilling

these requirements are listed in Section 1.3. Finally, the contributions of the thesis are summarized in Section 1.4.

In Chapter 2, the mathematical background required to understand the thesis is explained. It starts with a description of public key cryptography in Section 2.1, where ElGamal and Paillier cryptosystems are explained. Both cryptosystems offer a homomorphic encryption, as detailed in Section 2.2, which is widely used in electronic voting protocols. Section 2.3 is devoted to digital signature and blind signature schemes. The chapter is concluded with the definition and description of some zero-knowledge proofs in Section 2.4.

In Chapter 3, the mix-type paradigm is detailed in depth. It starts with a description of the paradigm in Section 3.1, where some well-known shuffling protocols are explained. The concept of optimistic mixing is introduced in Section 3.2, which leads to a novel design of optimistic mixing based on redundancy proofs presented in Section 3.3.

Next, in Chapter 4, the homomorphic tallying paradigm is described. In Section 4.1, the protocol involving an election following this paradigm is introduced. After that, a particular implementation of the paradigm based on array ballots is detailed in Section 4.2. Finally, in Section 4.3, a new approach for this implementation using shuffling protocols to prove the correctness of an array ballot is presented.

The remaining paradigm for secure remote voting, blind signature paradigm, is explained in Chapter 5. A description of the protocols involved in the paradigm is introduced in Section 5.1. A modification of these protocols using credentials to identify voters is explained in Section 5.2. Identifying a double voter in this paradigm has always been a challenge. In Section 5.3, the previous proposals which tried to achieve it are presented together with their respective security breaches. Finally, a new proposal able to identify double voters by using e-coins as voting credentials is presented in Section 5.4.

The last chapter is devoted to analysing the contributions to each paradigm presented in this thesis. Based on this analysis, the conclusions of the thesis are presented.

Chapter 2

Preliminaries

Remote voting systems provide security (see Section 1.2) by making use of public key cryptography. For instance, votes are transmitted encrypted so as to provide vote confidentiality. Voters have to authenticate themselves prior to ballot casting, which is usually achieved by making use of digital certificates and digital signatures. Remote voting systems are required to break the link between the ballots and the identity of the voters who cast them, prior to their decryption. This can be done in different ways such as shuffle and re-encrypt operations, homomorphic aggregation or blind signatures. Last but not least, with some paradigms the actors need to prove they have performed some action correctly. This has to be done without leaking any information that could jeopardize participants' privacy. Zero-knowledge proofs are an appropriate tool to that end.

In Section 2.1, a description of some public key cryptosystems is presented. By making use of encryption, a voter can cast her ballot in a confidential way through the Internet without leaking any information about its content.

All the cryptosystems introduced in Section 2.1 are *homomorphic*. This property is really useful in the design of remote voting systems. For this reason, this property and some of its applications are described in Section 2.2.

Public key cryptography also provides digital signature capabilities. Digital signatures, together with digital certificates, are widely employed for authentication and integrity preservation purposes. Section 2.3 is devoted to digital signature schemes.

Verifiability is a challenging requirement. Zero-knowledge proofs allow the parties involved in an election to prove they have acted honestly without providing any detail about the actions carried out. These proofs are introduced in Section 2.4.

2.1 Public key encryption

In a public key cryptosystem each user has two keys, one private and one public. The public key is made publicly available and is used for plaintext encryption or digital signature verification. The private key is kept secret and is used to decrypt ciphertexts or to create digital signatures.

Both keys are mathematically linked by a one-way trapdoor function so that it is computationally easy for an entity to generate a private / public key-pair, but it is computationally infeasible to determine the private key from its corresponding public key. Hence, a public key can be published since, from its knowledge, the private key cannot be obtained. Public key cryptography, unlike symmetric key cryptosystems, does not require a secure initial exchange of secret information (keys) between the parties.

The security of public key cryptosystems lies in the assumed intractability of some computational hard problems. The two most popular ones are the *integer factorization* [RSA78] and the *discrete logarithm* [ElG85] problems. The first one is the basis of cryptosystems like RSA or Paillier, while the second one leads to ElGamal cryptosystem (including its elliptic curve version). Some remote voting systems further require the employed cryptosystem to be probabilistic. This means that a given plaintext can be encrypted into many different ciphertexts. The RSA cryptosystem, in its basic form, does not provide this property, while Paillier and ElGamal do.

2.1.1 ElGamal cryptosystem

ElGamal [ElG85] is a probabilistic public key cryptosystem. As first proposed, its security holds on the assumed intractability of the *discrete logarithm problem* (DLP) over a subgroup G of the multiplicative group $\mathbb{F}_p^*$, p being prime. The problem is defined as follows. Given a generator g of G and an element $y \in G$, find an integer x such that $y = g^x$.

In order to make this problem computationally intractable, p has to be taken such that $p-1$ has a large prime factor q and G is taken as the order q subgroup of $\mathbb{F}_p^*$. This prevents the application of the Pohlig-Hellman method [PH78], which reduces the DLP over G into small instances over groups whose orders are factors of $p-1$.

ElGamal cryptosystem can be implemented over other groups such as the group of points of an elliptic curve defined over a finite field or the Jacobian of a hyperelliptic curve.

ElGamal cryptosystem is composed of the following procedures:

Setup and key generation

The setup is performed by choosing a prime p, and a generator g of a large prime order subgroup G of $\mathbb{F}_p^*$. With q being the order of G, a user $\mathcal{U}$ generates her private key by choosing a random integer

$x \in [1, \ldots, q-1]$. The public key is computed as $y = g^x \mod p$. Then, $\mathcal{U}$ can publish the system parameters p, q, g and her public key y. The private key x has to be kept secret.

Encryption
Message $m \in \mathbb{F}_p^*$ is encrypted into a ciphertext that only $\mathcal{U}$ will be able to decrypt. This is done by computing,

$$\text{Enc}_y(m) = (a, b) = (g^r \ , \ m \cdot y^r)$$

with $r \in_R [1, \ldots, q-1]$ being chosen at random. If the random value r is given as input it will be denoted as:

$$\text{Enc}_y(m, r) = (a, b) = (g^r \ , \ m \cdot y^r).$$

Decryption
An ElGamal ciphertext $c = (a, b)$ is decrypted, employing private key x, as follows:

$$\text{Dec}_x(c) = b/a^x = m.$$

Re-encrypt
An ElGamal ciphertext c can be re-encrypted in order to obtain a new ciphertext encrypting the same plaintext as follows:

$$\text{Rem}_{r'}(c) = (a \cdot g^{r'}, b \cdot y^{r'}).$$

There is an alternative way to obtain the cleartext of ciphertext c. This can be done if the random value r, generated at encryption, is revealed. First of all, the equality $g^r = a$ has to be checked. Next, compute

$$\text{Rev}_r(c) = b/y^r = m.$$

Notice that r is only known by the entity which generated the ciphertext.

2.1.2 Paillier cryptosystem

Paillier [Pai99] is a public key probabilistic cryptosystem whose security is based on the *decisional composite residuosity assumption*. This assumption states that, given a composite n and an integer z, it is hard to decide whether y exists such that

$$z \equiv y^n \mod n^2.$$

Paillier cryptosystem is composed of the following procedures:

Key generation
A user $\mathcal{U}$ creates a private/public key-pair by first choosing two large primes p and q such that $\gcd(pq, (p-1)(q-1)) = 1$. After that,

the values $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$ are computed. Next, a random integer $g \in \mathbb{Z}_{n^2}^*$ whose multiplicative order is divisible by n is generated. Finally, a value μ is computed as,

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

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

The tuple (n, g) is the public key while the tuple (λ, μ) is kept secret by $\mathcal{U}$.

Encryption

A message $m \in \mathbb{Z}_n$, is encrypted by computing,

$$\text{Enc}_{g,n}(m) = g^m \cdot r^n \mod n^2,$$

with $r \in_R [1, \ldots, n-1]$. If the random value r is given as input it will be denoted as:

$$\text{Enc}_{g,n}(m, r) = g^m \cdot r^n \mod n^2.$$

Decryption

A ciphertext c can be decrypted by computing,

$$\text{Dec}_{\lambda,\mu}(c) = L(c^\lambda \mod n^2) \cdot \mu \pmod n = m.$$

Re-encrypt

Any Paillier ciphertext c can be re-encrypted, or resmasked, to obtain a new ciphertext encrypting the same plaintext as follows:

$$\text{Rem}_{r'}(c) = c \cdot r'^n \mod n^2.$$

2.1.3 Elliptic ElGamal cryptosystem

An elliptic curve E over a prime finite field $\mathbb{F}_p$ is an algebraic curve given by an equation of the form

$$y^2 = x^3 + ax + b, \quad a, b \in \mathbb{F}_p,$$

with non-zero discriminant $4a^3 + 27b^2 \neq 0$, called reduced Weierstraß equation.

We denote by $E(\mathbb{F}_p)$ the set of points $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ that satisfy the curve equation, along with the point at infinity $\mathcal{O}$.

An addition operation can be defined over $E(\mathbb{F}_p)$ using the chord-tangent method. This operation endows the set $E(\mathbb{F}_p)$ with an abelian group structure in which $\mathcal{O}$ is the identity element. Considering this group law, a point $P \in E(\mathbb{F}_q)$ can be multiplied by a scalar d as follows

$$dP = \underbrace{P + \cdots + P}_{d \text{ times}}.$$

The point dP can be computed in an efficient way using the double-and-add algorithm.

The Elliptic ElGamal cryptosystem (EC-ElGamal) [Kob87, Mil86] is an analogue of the ElGamal cryptosystem using elliptic curves. Its security holds on the *elliptic curve discrete logarithm problem* (ECDLP). This problem is defined as, given two elliptic curve points P and Q such that $Q = dP$, find an integer d that solves the equation. This is a computationally hard problem.

The main advantage of EC-ElGamal compared with ElGamal cryptosystem defined over $\mathbb{F}_p$ is that the Index-Calculus [SS98] algorithm, which allows us to solve the DLP over $\mathbb{F}_p$ in subexponential time, cannot be employed over elliptic curve points. As a result, EC-ElGamal achieves equivalent security levels using shorter keys. For example, the security of ElGamal cryptosystem over $\mathbb{F}_p$ taking 1024 bit keys is equivalent to that of EC-ElGamal with 160 bit keys.

The algorithms composing EC-ElGamal are decribed below:

Setup and key generation

The cryptosystem setup first requires taking a prime p defining a field $\mathbb{F}_p$. Next, the two parameters a and b defining an elliptic curve E over $\mathbb{F}_p$ are chosen. The cardinality of the resulting elliptic curve should be divisible by a large factor q of the same size as p. Next, take an order q point $P \in E(\mathbb{F}_p)$.

A user $\mathcal{U}$ generates a keypair by choosing her private key d at random in $\{1, ..., q-1\}$. The corresponding public key is $Q = dP$.

Encryption

A point $V \in E(\mathbb{F}_p)$ is encrypted by computing:

$$\mathrm{Enc}_Q(V) = (A, B) = (rP, V + rQ),$$

with r being a random integer in $\{1, \ldots, q-1\}$. If the random value r is given as input it will be denoted as:

$$\mathrm{Enc}_Q(V, r) = (A, B) = (rP, V + rQ).$$

Decryption

User $\mathcal{U}$ can decrypt a ciphertext $C = (A, B)$, using her private key d. The cleartext V is obtained as follows:

$$\mathrm{Dec}_d(C) = B - dA = V.$$

Re-encrypt

The re-encrypt, or resmask, algorithm takes as input a ciphertext C

and a random value r' and outputs a new ciphertext encrypting the same plaintext of C. The algorithm proceeds as follows:

$$\text{Rem}_{r'}(C) = (A + r'P, B + r'Q).$$

The EC-ElGamal also provides a reveal algorithm. Given a ciphertext $C = (A, B)$ generated with a random value r, first check whether rP equals A and next compute

$$\text{Rev}_r(C) = B - rQ = V.$$

The reveal algorithm can only be used when the value r is known.

2.2 Homomorphic encryption

Some public key cryptosystems have an encryption function which offers a homomorphic property [FG07]

$$\text{Enc}(m_1) \oplus \text{Enc}(m_2) = \text{Enc}(m_1 \odot m_2),$$

in which $\oplus$ is a binary operation defined over the set of ciphertexts and $\odot$ is a binary operation defined over the cleartext space. Such a property establishes that the result of $\text{Enc}(m_1) \oplus \text{Enc}(m_2)$ is an encryption of $m_1 \odot m_2$.

A cryptosystem with a homomorphic property is called a homomorphic cryptosystem. Depending on the operation $\odot$, the homomorphic property can be additive or multiplicative.

Paillier [Pai99] is an additive homomorphic cryptosystem. As seen above, its encryption algorithm computes a ciphertext as,

$$\text{Enc}(m) = g^m \cdot r^n \mod n^2,$$

for some random r. In this case, the homomorphic operation $\oplus$ is the ciphertext (modular) product. Notice that,

$$\text{Enc}(m_1) \cdot \text{Enc}(m_2) = g^{m_1+m_2} \cdot (r_1 r_2)^n = \text{Enc}(m_1 + m_2).$$

ElGamal is a multiplicative homomorphic cryptosystem. Let us recall that a message m is encrypted as:

$$\text{Enc}(m) = (g^r, m \cdot y^r).$$

The homomorphic operation $\oplus$ is the component-wise product of ciphertexts, here represented as $\otimes$, and the operation $\odot$ performed on the cleartext is the multiplication in $\mathbb{Z}_p^*$. Then,

$$\text{Enc}(m_1) \otimes \text{Enc}(m_2) = (g^{r_1+r_2}, m_1 \cdot m_2 \cdot y^{r_1+r_2}) = \text{Enc}(m_1 \cdot m_2).$$

ElGamal can be modified in order to provide an additive homomorphic property. This is achieved by encrypting g^m:

$$\text{Enc}_{add}(m) = \text{Enc}(g^m) = (g^r, g^m \cdot y^r),$$

Then, the ciphertext componentwise product behaves as follows:

$$\text{Enc}_{add}(m_1) \otimes \text{Enc}_{add}(m_2) = (g^{r_1+r_2}, g^{m_1+m_2} \cdot y^{r_1+r_2}) = \text{Enc}_{add}(m_1 + m_2).$$

The elliptic ElGamal cryptosystem provides a point addition homomorphism. A point $V \in E(\mathbb{F}_p)$ is encrypted as:

$$\text{Enc}(V) = (rP, V + rQ).$$

The homomorphic operation $\oplus$ is the component-wise ciphertext point addition, and the resulting operation $\odot$ on cleartexts is the point addition. It results in the following homomorphic property:

$$\text{Enc}(V_1) \oplus \text{Enc}(V_2) = ((r_1 + r_2)P, V_1 + V_2 + (r_1 + r_2)Q) = \text{Enc}(V_1 + V_2).$$

EC-ElGamal can also be adapted to provide an integer additive homomorphism. Given an integer m, compute $V = mP$ and encrypt V. Thus,

$$\text{Enc}(mP) = (rP, mP + rQ),$$

In this case,

$$\text{Enc}(m_1P) \oplus \text{Enc}(m_2P) = \text{Enc}((m_1 + m_2)P).$$

Notice that the re-encryption algorithm of each cryptosystem can be viewed as a homomorphic operation combining two ciphertexts, one of them encrypting the neuter element.

2.3 Digital Signature

A digital signature is a concept introduced in [DH76] which provides the following security properties for the signed message:

- *Authentication*: the reader of a signed message knows it was signed by the claimed signer.
- *Non-repudiation*: the signer cannot deny having signed a message.
- *Integrity*: If a signed message is modified, this will be detected.

2.3.1 Hash functions

A hash function $\mathcal{H}$ maps digital information of arbitrary length into a fixed length string. When used for cryptographic purposes, a hash function must provide the following properties:

- *Determinism*: Given an input message, its hash value is always the same.
- *Uniformity*: Each hash value in the output range can be generated with almost the same probability.
- *Non-invertibility*: It is infeasible to generate a message from its hash.
- *Efficiency*: Hash computation is a lightweight procedure.
- *Collision resistance*: Finding two different messages resulting in the same hash is hard.

Hash functions are a fundamental tool for digital signature. As we will see next, they are employed to reduce the size of the data to be signed. A digital signature on message m is computed by signing its hash digest $\mathcal{H}(m)$.

2.3.2 Digital signature algorithms

A digital signature scheme consists of three algorithms:

1. Key generation: This algorithm generates a private / public keypair. Let x and y denote the private and public keys, respectively.
2. Sign: Given a message m, this algorithm generates its signature s under private key x:
$$s = \text{Sign}_x(m).$$
3. Verify: Given a message m, its signature s and a public key y, this algorithm determines whether s is a digital signature on m computed using the private key related to x:
$$\text{Ver}_y(m, s) = \text{true/false}.$$

As an example, we show how the RSA [RSA78] digital signature works:

1. Key generation: Generate two large primes p and q of the same size. After that, compute $n = pq$ and $\varphi(n) = (p-1)(q-1)$. Finally, take a random integer $e \in \{2, \ldots, \varphi(n) - 1\}$ and compute its inverse $d = e^{-1} \mod \varphi(n)$.

   The public key is the tuple (n, e) and the private key is d.

2. Sign: The hash of message m, $\mathcal{H}(m)$, is signed under private key d as follows:
$$s = \text{Sign}_d(\mathcal{H}(m)) = \mathcal{H}(m)^d \mod n.$$

3. Verify: A signature s on message m is verified by algorithm $\text{Ver}_{(n,e)}(m, s)$ which checks whether the following equation holds:
$$s^e \mod n = \mathcal{H}(m).$$

Both the signer and the verifier should agree on the hash function $\mathcal{H}$ to be used.

2.3.3 Blind signature protocols

A blind signature protocol involves two parties, a message owner and a signer. As a result, the message owner obtains a digital signature computed by the signer over her message while the signer obtains no information about the signed message. A blind signature protocol is composed of the following procedures:

1. Key generation: The signer generates a private / public key pair, x, y.

2. Blind: This algorithm disguises the original message m using a random parameter r generating $m' = Blind_r(m)$ as a result.

3. Blindly Sign: The disguised message m' is sent to the signer who computes a digital signture on it. The result, $s' = \text{Sign}_x(m')$, is sent to the message owner.

4. Unblind: The digital signature s' over m' is unblinded in order to obtain a digital signature s over m. This algorithm receives s' and r as input and generates $s = \text{Unblind}_r(s')$ as output. The result is the same as if the signer had computed $s = \text{Sign}_x(m)$.

5. Verify: Given a message m, its signature s and a public key y, the algorithm checks whether s is a correct signature of m under public key y:
$$\text{Ver}_y(m, s) = \text{true/false}.$$

The RSA blind signature algorithm is detailed in the following lines:

1. Setup: Generate an RSA keypair, d and (n, e).

2. Blind: The hash digest of message m, $\mathcal{H}(m)$, is blinded using a random factor r by computing
$$m' = Blind_r(\mathcal{H}(m)) = \mathcal{H}(m) \cdot r^e \mod n.$$

3. Blindly Sign: The blinded message m' is signed as:

$$s' = \text{Sign}_d(m') = (m')^d = \mathcal{H}(m)^d \cdot r \mod n.$$

4. Unblind: The digital signature s' over m' is transformed into s as follows:

$$s = \text{Unblind}_r(s') = s' \cdot r^{-1} \mod n.$$

The resulting signature is $s = \mathcal{H}(m)^d = \text{Sign}_d(\mathcal{H}(m))$.

5. Verify: The signature s can be verified by means of algorithm

$$\text{Ver}_{(n,e)}(\mathcal{H}(m), s).$$

2.4 Zero-knowledge proofs

Proving that one possesses certain knowledge is in most cases trivial if one is allowed to simply reveal that knowledge. The challenging task is to prove such knowledge without revealing it. A zero-knowledge proof [GMR89] is a method by which a prover can prove to a verifier that a given statement is true, without conveying any additional information apart from the fact that the statement is indeed true.

A zero-knowledge proof (ZKP) must satisfy three properties:

- *Completeness*: if the statement is true, the verifier will be convinced of this fact by a prover.
- *Soundness*: if the statement is false, no cheating prover can convince the 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.

2.4.1 ZKP for discrete logarithm equality

The ZKP for discrete logarithm equality was first designed as a digital signature verification algorithm by Chaum and Pedersen [CP93]. It can be used as a tool to provide a verifiable decryption of ElGamal ciphertexts.

The proof states that, given four elements (g, y, m, z), the prover knows a value x such that $g^x = y$ and $m^x = z$. That is, $\log_g y = log_m z$. In order to do that, the proof proceeds (in its non-interactive form) as follows:

Prover

- Choses s at random and computes the tuple $(a', b') = (g^s, m^s)$.
- Generates $h = \mathcal{H}(m, z, a', b')$.
- Computes $r = s + hx$.
- Sends (a', b', r) to the verifier.

Verifier

- Computes $h = \mathcal{H}(m, z, a', b')$.
- Checks that $g^r = a'y^h$ and $m^r = b'z^h$.

Let $c = (a, b)$ be an ElGamal ciphertext. The only party able to decrypt it is the one in possession of the secret key x. This party is able to prove in zero-knowledge that a message m is really obtained from the decryption of c without revealing any information about the secret key. This is done by proving in zero-knowledge that $\log_g y = \log_a(b \cdot m^{-1})$. In such a case, we say that c has been verifiably decrypted.

2.4.2 ZKP for message lies in set

Cramer et al. [CDS94] presented an approach to construct proofs of partial knowledge. This approach has been implemented as range proofs. These proofs permit us to demonstrate that the cleartext encrypted in a ciphertext belongs to a given set. Next, we detail its implementation for range proofs of ciphertexts encrypted under EC-ElGamal.

Prover

Let $\mathcal{P}$ be a prover who wants to prove that an EC-ElGamal ciphertext $C = (A, B)$ is encrypting a point S_k which lies in the set $S = \{S_1, S_2, \ldots, S_k\}$. $\mathcal{P}$ would proceed as follows:

1. $\mathcal{P}$ randomly generates w_j, u_j, x for $j = 1, 2, \ldots, k-1$.
2. $\mathcal{P}$ computes

$$\begin{aligned} A'_j &= w_j P + u_j A \quad \text{for} j \neq k, \\ B'_j &= w_j Q + u_j (B - S_j) \quad \text{for} j \neq k, \\ A'_k &= xP, \\ B'_k &= xQ. \end{aligned}$$

3. $\mathcal{P}$ computes $chall = \mathcal{H}(A', B')$, A' and B' being the concatenation of the values A_j and B_j for $1 \leq j \leq k$, respectively.

4. $\mathcal{P}$ sends:

$$w_1, w_2, \ldots, w_k \quad u_1, u_2, \ldots, u_k,$$

with

$$u_k = chall - \sum_{j=1}^{k-1} u_j,$$
$$w_k = x - u_k r.$$

Notice that r is the random integer used in the generation of C and it is only known by the $\mathcal{P}$.

Verifier

The verifier knows $C = (A, B)$ and the set S. When she receives

$$w_1, w_2, \ldots, w_k \quad u_1, u_2, \ldots, u_k,$$

she proceeds with the following checks:

1. The verifier checks that $\mathcal{H}(A', B') = \sum_{j=1}^{k} u_j$.
2. For each j, $1 \leq j \leq k$, the verifier checks that

$$A'_j = w_j P + u_j A, \quad B'_j = w_j Q + u_j (B - S_j).$$

If all the checks are satisfied, the verifier is convinced that C is an encryption of a point in the set S.

Chapter 3

Mix-type paradigm

This chapter begins with a description of the mix-type remote voting paradigm. The main idea and its properties are explained in Section 3.1. Mix-type remote voting offers privacy by breaking the link between the ballots and the voters who cast them through a mixing process whose correctness has to be proven in zero-knowledge. One of the approaches is the so-called *optimistic mixing*, a description of which is given in Section 3.2. After that, in Section 3.3, we present a new proposal for optimistic mixing which was published in [MMS11].

3.1 Paradigm description

The main phases of an electronic election have been described in Section 1.1. As has been explained, some paradigms require additional entities and include additional phases. The mix-type paradigm includes an additional phase, called *mixing*, in which the link between each ballot and its caster is broken by a set of *mixing authorities*. Usually, a mix-type paradigm voting scheme is composed of the following four phases:

1. **Setup:** A public key cryptosystem is chosen and the election keys are generated by the vote tallying authorities. Next, the election public key is published on a publicly accessible bulletin board. At the same time, the electoral roll authority publishes the electoral roll so that it can be checked. The electoral roll includes information required to verify the public key of each eligible voter. This information could be a digital certificate of the voter's public key or simply the public key of the accepted certificate authorities. The list of candidates as well as the instructions for voters are also published.

   As a result of this phase, the bulletin board contains the election public key, the electoral roll and the candidates list. The bulletin board permissions are then set so that these data can no longer be modified.

2. **Vote casting:** During this phase, a voter generates a message describing her choice and encrypts it under the election public key. The resulting ciphertext is then digitally signed using the private key associated with her digital certificate. After that, the ballot and its signature are sent to the vote collecting entity through the Internet.

   Upon receiving a ballot, the vote collecting entity must authenticate the ballot caster and check that she appears in the electoral roll. This is done by checking the validity of the digital signature using the voter's digital certificate. If the caster is an eligible voter and has not cast a ballot before, then the vote collecting authority publishes the ballot on the bulletin board so that any entity can perform the same checks.

   The vote casting phase ends when the voting period expires or when all the eligible voters have cast their ballots.

3. **Mixing:** During this phase, the so-called *mixing authorities* are required. These authorities are organized in a sequential manner such that the first one takes as input the ballots published on the bulletin board. These ballots are mixed (permuted and re-encrypted). The second mixer then takes as input the output of the first authority and performs the same operation. This process is repeated for each mixer. In the end, a set of mixed ballots is obtained. Each mixing authority publishes its result together with a zero-knowledge proof proving that its output really corresponds to a mixing of its input. In this way, any entity can check its correctness.

4. **Tallying:** The resulting set of mixed ballots is verifiably decrypted by the the vote tallying authorities. Once the ballots have been decrypted, the election result is published on the bulletin board.

Since ballots are individually decrypted, there is no restriction on its format or coding. Hence, the range of messages that can be encrypted is only bounded by the cleartext message length. As a consequence, the paradigm can be used in elections in which vote coding is rather complicated. If, after decrypting it, some vote is found to have been badly generated, it can simply be discarded.

The most complex and time-consuming part of this paradigm is the generation and validation of the zero-knowledge proof of correct mixing. The next section is devoted to this part.

3.1.1 Mixing processes

Privacy is a fundamental requirement in any election. Vote encryption is necessary but it is not enough to provide privacy. Encrypting a vote provides privacy at vote transmission but the link between a voter and her choice would be revealed after ballot decryption.

In mix-type remote voting systems, a mixing process is run after all the ballots have been received at the ballot collection authority. Such a process takes as input a set of ballots encrypted under some public key cryptosystem offering a re-encryption operation. A mixing process includes two operations: ciphertext shuffling using a random permutation π and the re-encryption of the permuted ciphertexts. Re-encryption is needed so that the mixed ciphertexts cannot be related to the original non-mixed ones. A mixing operation generates as output the new set of ciphertexts together with a zero-knowledge proof of correctness.

Such a mixing process is depicted in Figure 3.1 in which a set of ballots $C_{v_1}, \ldots, C_{v_n}$ is mixed. As a result, we obtain a set of mixed ciphertexts $C'_1, \ldots, C'_n$ together with a zero-knowledge proof of correctness.

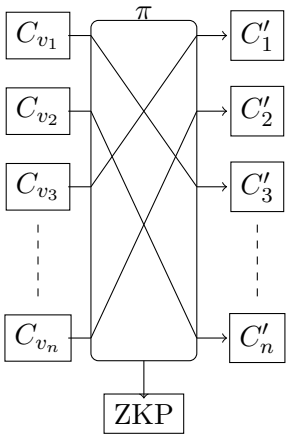

Figure 3.1: Verifiable mixing process.

The idea of using a shuffle operation to break the binding between each ballot and its caster was first proposed in [Cha81]. In that proposal, a *decryption mix-net* was employed. A decryption mix-net requires the ballots to be encrypted under several public keys, one for each mixing authority (or mixer). Once all the ballots have been cast, the first mixer decrypts the ballots using its private key, so that its encryption layer is removed. Then, the mixer shuffles the results and sends the result to the next mixer. The next mixers proceeds in the same manner. The last mixer output corresponds to the cleartext votes which can now be tallied. Unfortunately, the correctness of such a mix-net cannot be proven and therefore the integrity property is not guaranteed. Jakobsson, Juels, and Rivest [JJR02] proposed a different technique to address that problem and gave heuristic arguments showing that, in their proposal, it would be difficult to change more than a small number of ciphertexts without being noticed. That technique, called Randomized Partial Checking (RPC) [PG10], requires several mixers, as in Chaum's

proposal. In this case, if more than just a few ciphertexts are incorrectly processed by a mixer, the bad behaviour will be noticed with high probability. That scheme and several of its implementations [CEC+08, CCM08], which have been used in real elections, have recently been proven insecure in [KW13] against Pfitzmann's Attack [Pfi95, PP90].

A slightly different approach requiring homomorphic cryptosystems was proposed by Park, Itoh and Kurosawa [PIK94]. That proposal was the first use of a *re-encryption mix-net*. In that proposal, the mixers collectively generate the election public key so that the private key required for decryption is distributed among them. During the mixing phase, the mixers sequentially shuffle the ballots and re-encrypt them. Finally, the resulting ciphertexts are jointly decrypted. The first universally verifiable mix-net was a re-encryption mix-net designed by Sako and Kilian [SK95]. In that proposal, the senders can verify that the entire mixing has been performed correctly. From then on, the design of new mix-nets with efficiency improvements has been recurrent in the bibliography. It is important to highlight the performance improvement of the proposals by Neff [Nef01] and Furukawa and Sako [FS01]. Those proposals employ zero-knowledge proofs (ZKP) of correct mixing which outperform the previous cut-and-choose proposals. Currently, the proposals of Telerius and Wikström [TW10] and Bayer and Groth [BG12] are the most secure and efficient mixing proposals followed by Peng's mixnet [Pen11a], which is more efficient in cost but weaker in terms of security.

A mixing procedure has to meet the following properties:

- *Privacy:* The permutation π employed to shuffle the ballots remains secret.
- *Correctness:* The plaintexts of the output ballots equal the plaintexts of the input ones.
- *Public verifiability:* The *correctness* property can be verified by any external entity. Each mixer can publicly prove that it performed the mixing process correctly.

The resulting permutation π applied to the ballots is the composition of the permutations applied by each mixing authority. In this way, π remains secret as long as at least one of the mixing authorities does not reveal its permutation. Hence, the mixing authorities have to be chosen so that each voter can, at least, trust one of them.

Regarding public verifiability, ensuring the correctness of the election without leaking sensitive information is mandatory. This is achieved by making use of the so-called *zero-knowledge proofs of mixing*.

3.1.2 Zero-knowledge proofs of mixing

Generally speaking, a zero-knowledge proof of mixing is generated by a procedure that receives as input a set of input ciphertexts $C_1, \ldots, C_n$, the set of mixed ciphertexts $C'_1, \ldots, C'_n$, the permutation π and the re-encryption factors $r_1, \ldots, r_n$. This procedure generates as output some data that demonstrate that the set of mixed ciphertexts was obtained from the set of received ones after performing a correct mixing operation.

As introduced in Section 2.4, a zero-knowledge proof has three main properties: completeness, soundness and zero-knowledge. These properties are perfectly suited for the requirements of a proof of mixing. The completeness and soundness properties guarantee the integrity of the election. The zero-knowledge property is required for privacy while allowing public verifiability.

Several proposals of zero-knowledge proofs of mixing exist. Some of them achieve better performance at the cost of reducing the randomness of the permutation. The security of these electronic voting systems is weaker than that of schemes using a completely random permutation, but they are strong enough. An example of such an approach was proposed by Peng [Pen11a]. During the mixing operation, the ballots are divided into several groups of the same size. Then, each group is shuffled and re-encrypted using the same parameters. The zero-knowledge proof of correct mixing is generated for one group and then, by using batch techniques, the proof is extended to prove the correctness of the mixing applied to the other groups. This is possible because the proof bases its correctness on the fact that given a set of integers s_i and s'_i, for $i = 1, \ldots, n$, it can demonstrate (in zero-knowledge) knowledge of a set of values t_i and t'_i, for $i = 1, \ldots, n$, satisfying that:

$$\sum_{i=1}^{n} \mathrm{Dec}(C_{v_i}) \cdot s_i = \sum_{i=1}^{n} \mathrm{Dec}(C'_i) \cdot t_i,$$

$$\sum_{i=1}^{n} \mathrm{Dec}(C_{v_i}) \cdot s'_i = \sum_{i=1}^{n} \mathrm{Dec}(C'_i) \cdot t'_i,$$

$$\sum_{i=1}^{n} \mathrm{Dec}(C_{v_i}) \cdot s_i \cdot s'_i = \sum_{i=1}^{n} \mathrm{Dec}(C'_i) \cdot t_i \cdot t'_i,$$

being $\mathrm{Dec}(C_{v_i})$ the decryption function of the Elliptic ElGamal cryptosystem receiving as input the ciphertext C_{v_i}. In [Pen11a], it is proven that the existence of these integers t_i and t'_i, for $i = 1, \ldots, n$, implies, with high probability, that the set $\{\mathrm{Dec}(C'_1), \ldots, \mathrm{Dec}(C'_n)\}$ is a permutation of $\{\mathrm{Dec}(C_{v_1}), \ldots, \mathrm{Dec}(C_{v_n})\}$.

As can be seen, [Pen11a] proves the correctness of a mixing operation by demonstrating that a certain condition is met. In this case, the completeness

and soundness properties are obtained from this condition and not from the applied permutation and re-encryption. As a consequence, this proof states that the correctness property is satisfied with high probability.

A different proof is proposed by Hoogh et al. [dHSV09]. In that case, the permutation is a rotation. They designed a proof which makes use of the Discrete Fourier Transform (DFT). Based on this proof, Telerius and Wikström [TW10] proposed a zero-knowledge proof of correct mixing without restrictions on the permutation as occurs in the well known proposals by Neff [Nef01] and Furukawa and Sako [FS01]. That adaptation opened the window to more efficient shuffling proofs that can be combined with the offline computation techniques introduced in [Wik09]. This allows the proof generation to be divided into two phases. During the first one, the prover commits to the permutation applied for shuffling. This can be precomputed offline, reducing the amount of computations required during the online part. After that, the prover engages in an online proof in which it proves that, during the shuffling and re-encryption of the ballots, he used the permutation he previously committed to. He also proves knowledge of the random values used for re-encrypting the ballots received.

Following the same approach, Bayer and Groth [BG12] proposed a proof with a better performance in terms of speed and memory space. As in Wikström's proposal, it first commits to a given permutation and later engages in proving knowledge of the re-encryption values, described in the paper as multi-exponentiation and product arguments. More precisely, given the set of ciphertexts $C_1, \ldots, C_n$ received,the set of mixed ciphertexts $C'_1, \ldots, C'_n$, the permutation π and the re-encryption factors $r_1, \ldots, r_n$, it proves knowledge of π and $r_1, \ldots, r_n$ such that:

$$C'_i = \mathrm{Enc}_{pk}(1, r_i) \cdot C_{\pi(i)},$$

where $\mathrm{Enc}_{pk}(1, r_i)$ is the cryptosystem's encryption function with public key pk of the neuter element 1. Notice that both [BG12] and [TW10] guarantee correctness of the shuffle. This is because the zero-knowledge proof is generated from the permutation and the re-encryption values. This means that a remote voting platform implementing those mixing procedures guarantees the integrity property.

In Table 3.1 we can see the general improvements to the cost of the zero-knowledge proofs. It is interesting to notice the small difference between the cost of the different solutions. In this table the cost of [TW10] does not take into account the improvements described in [Wik09]; for this reason, the performance is quite low. However, with the appropriate implementation, the cost is similar to [BG12].

Table 3.1: Comparison of ZKP proposals

ZKPs	Prover Exponentiations	Verifier exponentiations
[FS01]	$8n$	$10n$
[Nef01]	$8n$	$10n$
[TW10]	$9n$	$11n$
[GL07]	$5n$	$4n$
[BG12]	$2(\log m)n$	$4n$
[Pen11a]	$4n$	$2n$

3.2 Optimistic mixing

The optimistic mixing approach considers that the mixers will rarely misbehave. The idea is to provide an efficient protocol which is able to prove the mixing's correctness when all the entities behaved correctly, but which is also capable of detecting a liable mixer in case of misbehaviour. In that case, a protocol is run in which the mixers are asked to provide a fully secure zero-knowledge proof of correct mixing (see Section 3.1.2). A mixer unable to provide it will be assumed to have cheated.

The first proposal of optimistic mixing was presented by Jakobsson in [Jak98], with a privately convincing mix-net in which mixers can only convince themselves of correctness. The efficiency of the proposal was better than any other existing mix-net, but was based on less strict security definitions. Jakobsson's proposal was proven insecure in [DK00], but the idea has later been used in other proposals [JJ01, GZB+02, SMPP10].

The schemes proposed by Golle et al. [GZB+02] and Sebé et al. [SMPP10] use a technique to prove the correctness of the mixing using the *proof of product with redundancy* approach. With these schemes, participants add some redundancy to their votes prior to encryption. Correctness of mixing is proven by performing two checks. The first check consists of homomorphically aggregating (in a multiplicative way) the ballots received into a single ciphertext that will be proven to contain the same cleartext as the aggregation of mixed ciphertexts. Next, redundancy of each vote is checked at vote decryption. With current proposals, votes are encrypted by means of a double layer system so that redundancy can be checked after removing the first encryption layer. In this way, in case of redundancy failure, no cleartext vote has yet been revealed.

The main advantage of this paradigm is that, when the mixing party is composed of several mixing elements, the proof of correctness is applied only between the ballots received and the output of the last mixing authority. In this way, the system can accommodate a large amount of mixing authorities without increasing the cost of the proof.

3.2.1 Golle et al. proposal

In [GZB+02], ballots are submitted by sending the following tuple of ElGamal ciphertexts:

$$(\mathrm{Enc}_y(G), \mathrm{Enc}_y(M), \mathrm{Enc}_y(\mathcal{H}(G, M)))$$

with $(G, M) = \mathrm{Enc}_y(m)$ and $\mathcal{H}$ being a hash function.

At the end of the vote collecting phase, the ballot collection authority has stored a set of ballots

$$L = \{(\mathrm{Enc}_y(G_i), \mathrm{Enc}_y(M_i), \mathrm{Enc}_y(H_i))\}_i.$$

These tuples are then mixed so that a set of mixed ballots is obtained

$$L' = \{(\mathrm{Enc}_y(G'_i), \mathrm{Enc}_y(M'_i), \mathrm{Enc}_y(H'_i))\}_i.$$

The proof of correct mixing consists of proving that

$$\prod_i G_i = \prod_i G'_i, \quad \prod_i M_i = \prod_i M'_i, \quad \prod_i H_i = \prod_i H'_i,$$

and also checking that, for each i,

$$\mathcal{H}(G'_i, M'_i) = H'_i.$$

If all these checks are satisfied, ciphertexts (G'_i, M'_i) are finally decrypted, obtaining the cleartext votes v_i.

Wikström presented an attack against this system in [Wik04]. It was proven that the tracing procedure proposed in [GZB+02] included several vulnerabilities. The proposed attack could be performed by a malicious participant who could cast a ballot with a non-matching redundancy, forcing the execution of the vulnerable tracing procedure. Wikström also describes some vulnerabilities involving dishonest mixing authorities able to bypass honest ones.

3.2.2 Sebé et al. proposal

The proposal by Sebé et al. [SMPP10] follows a similar idea, but votes are composed as follows:

1. Encrypt vote v_i using the ECIES elliptic curve cryptosystem $V_i = \mathrm{Enc}_Q(v_i)$.
2. Generate message $m_i = V_i||\mathcal{H}(V_i)||b_i$, where b_i is chosen so that m_i is a quadratic residue of $\mathbb{F}_p^*$.
3. Encrypt m_i under ElGamal encryption scheme as $C_i = \mathrm{Enc}_y(m_i)$.

The list of votes collected $L = \{C_i = \text{Enc}_y(m_i)\}_i$ consists of single ElGamal ciphertexts. They will be mixed, resulting in $L' = \{C'_i = \text{Enc}_y(m'_i)\}_i$. Correctness of mixing can be verified by checking that

$$\prod_i m_i = \prod_i m'_i$$

and also checking that, for each i, parsing $m'_i = V_i||H_i||b_i$ then $\mathcal{H}(V_i) = H_i$. If this is the case, ECIES ciphertexts V_i will be decrypted. Note that [SMPP10] takes advantage of the fact that ECIES ciphertexts can be directly encrypted in a single ElGamal cryptogram.

The family of attacks involving dishonest mixing elements that bypass honest ones from [Wik04] is avoided in [SMPP10], using a technique in which each mixer adds some dummy ballots.

In this proposal, as in the previous one, a malicious participant could send an invalid ballot with a non-matching redundancy. This attack would not compromise the privacy of the system; however, the vote opening phase would have to be interrupted and the whole election would have to be carried out again.

3.3 New proposal

A new mix-type voting system that follows the *proof of product with redundancy* paradigm will be presented next . The new proposal [MMS11] includes the following contributions:

- A technique for composing encrypted votes with redundancy whose correct generation can be proven in zero-knowledge. This prevents attacks by malicious participants sending badly composed votes or message relation attacks.
- A dummy ciphertext addition (and subsequent removal) method aiming to detect honest mixing element bypassing.
- A technique for checking redundancy after ballot mixing without information leakage in case of failure. If this was the case, the dishonest mixing element(s) would be traced and, after removing it/them, the process would be able to continue.

The resulting system overcomes the security and operational drawbacks of previous proposals [GZB+02, SMPP10] and shows that the *proof of product with redundancy* paradigm is viable.

The new scheme is composed of four phases beginning with a preliminary *setup* in which the cryptosystem parameters are generated. The second one is the *vote casting* phase. It is composed of a procedure for ballot generation

and submission and a zero-knowledge proof in which the participant proves that the ballot submitted was properly generated. Next, the *mixing* phase includes the mixing procedure with dummy ballot addition and the subsequent method for tracing and removing dummies. Finally, the *tallying* phase checks that the whole procedure has been performed correctly. After that, the ballots are verifiably decrypted and the election results are gathered.

3.3.1 Setup

This is a preliminary phase in which the key storage trusted party generates the required cryptographic material. First of all, two multiplicative cyclic groups for two ElGamal cryptosystems are generated:

1. Generate three large primes q, p and p' satisfying that $p = 2q + 1$ and $p' = 2p + 1$ (q, p, p' are a Cunningham chain of length 3).
2. Define an element $g \in \mathbb{F}_p^*$ that is a generator of the order q of the multiplicative subgroup of $\mathbb{F}_p^*$, $G = \langle g \rangle$.
3. Define an element $g' \in \mathbb{F}_{p'}^*$ that is a generator of the order p of the multiplicative subgroup of $\mathbb{F}_{p'}^*$, $G' = \langle g' \rangle$.

Next, two private/public key pairs are created:

1. Generate and store secret key $x \in [1, q-1]$ and its related public key $y = g^x$.
2. Generate and store a secret key $x' \in [1, p-1]$ and its related public key $y' = g'^{x'}$.

Parameters q, p, p', g, g' and public keys y, y' are made public. All these parameters must be digitally certified by some trusted authority.

3.3.2 Vote casting

A participant $\mathcal{P}$ aiming to submit vote m (encoded as an integer), generates and sends the encrypted redundant vote in the following way:

1. Generate $v = m \cdot 2^l + t$ where t is a random l bits integer so that $v \in G$.
2. Generate the following tuple:

$$(C, R) = \left(\mathrm{Enc}_y(v, r), \mathrm{Enc}_{y'}(g'^v, r')\right) = \left((g^r, v \cdot y^r), (g'^{r'}, g'^v \cdot y'^{r'})\right),$$

   with $r \in [1, q-1]$ and $r' \in [1, p-1]$ being chosen at random.
3. Send (C, R) (digitally signed) to the ballot collection authority.

Upon receiving (C, R) and its signature, the ballot collection authority will check that the participant appears in the electoral roll and that she has not voted yet. Next, the participant will be required to prove in zero-knowledge that the vote has been properly composed by means of the procedure described below.

Proving redundancy correctness

Given an encrypted redundant vote (C, R) submitted by participant $\mathcal{P}$ (the prover), next we shown how $\mathcal{P}$ convinces the ballot collection authority $\mathcal{V}$ (the verifier) of its correct composition, *i.e.* $\mathcal{P}$ proves in zero-knowledge that

$$g'^{\mathrm{Dec}_x(C)} = \mathrm{Dec}_{x'}(R).$$

1. $\mathcal{P}$ generates $v' \in G$, $r'' \in [1, q-1]$ and $r''' \in [1, p-1]$ at random and sends
$$C' = \mathrm{Enc}_y(vv', r'') \quad \text{and} \quad R' = R^{v'} \cdot \mathrm{Enc}_{y'}(1, r''')$$
to $\mathcal{V}$.

2. $\mathcal{V}$ generates a random challenge bit *chall* that is sent to $\mathcal{P}$.

3. If $chall = 0$:

   (a) $\mathcal{P}$ sends $z = r'' - r \pmod q$ and $d = r'''$ to $\mathcal{V}$.

   (b) $\mathcal{V}$ computes $\hat{v} = \mathrm{Rev}_z(C' \cdot C^{-1})$ and checks whether
   $$R' = R^{\hat{v}} \cdot \mathrm{Enc}_{y'}(1, d).$$
   The multiplication operation of C' and C^{-1} is component-wise and the inverse of $C = (a, b)$ is $C^{-1} = (a^{-1}, b^{-1})$.

   If $chall = 1$:

   (a) $\mathcal{P}$ sends $z = r''$ and $d = r'v' + r''' \pmod p$ to $\mathcal{V}$.

   (b) $\mathcal{V}$ computes $\hat{vv'} = \mathrm{Rev}_z(C')$ and checks whether
   $$g'^{\hat{vv'}} = \mathrm{Rev}_d(R').$$

The previous proof is zero-knowledge. This is proven by showing that there exists a simulator that can produce a transcript that "looks like" an interaction between the honest prover and the verifier. An iteration of the proof for an arbitrary tuple (C, R) can be simulated if the challenge bit is known in advance as detailed below.

1. Generate $k \in G$, $r'' \in [1, q-1]$ and $r''' \in [1, p-1]$ at random.

2. If the challenge is $chall = 0$:

- Compute $C' = \text{Enc}_y(k, r'') \cdot C$ and $R' = R^k \cdot \text{Enc}_{y'}(1, r''')$.

If the challenge is $chall = 1$:

- Compute $C' = \text{Enc}_y(k, r'')$ and $R' = \text{Enc}_{y'}(g'^k, r''')$.

3. Let $z = r''$ and $d = r'''$.

It is easy to see that the tuple $(C', R', chall, z, d)$ is a good simulation of the proof (it satisfies the verifier's checking). In the previous protocol, a malicious prover could cheat if he correctly guessed the value of the challenge bit *chall* sent by the prover in advance. Since this happens with probability $1/2$, repeating the previous protocol t times reduces the cheating probability to $1/2^t$.

3.3.3 Mixing

Let $\mathcal{L} = \{(C_i, R_i)\}_{0 \le i < n}$ be the set of encrypted votes collected by the ballot collection authority once the vote reception period has ended. The first mixing element adds some dummy votes and next its content is sequentially shuffled and re-encrypted by each mixing element. The procedure is detailed below.

1. The first mixing element, ME_1, generates s dummy votes

$$\{(\hat{C}_j, \hat{R}_j) = (\text{Enc}_y(1, t_{1,j}), \text{Enc}_{y'}(1, t'_{1,j}))\}_{0 \le j < s}$$

for some randomly generated values $t_{1,j} \in [1, q-1]$ and $t'_{1,j} \in [1, p-1]$.

2. ME_1 appends the s dummies to the list of received votes, so that they will be referred to as

$$(C_{n+j}, R_{n+j}) = (\hat{C}_j, \hat{R}_j), \quad \text{for} \quad 0 \le j < s.$$

3. ME_1 generates a secret $n + s$ elements permutation π_1 and a random set $\{r'_{1,i}, r''_{1,i}\}_{0 \le i < n+s}$ and computes a new list of shuffled and re-encrypted ciphertexts (including dummies) as

$$\mathcal{L}^{(1)} = \{(C^{(1)}_{\pi_1(i)}, R^{(1)}_{\pi_1(i)})\} = \{(C_i \cdot \text{Enc}_y(1, r'_{1,i}), R_i \cdot \text{Enc}_{y'}(1, r''_{1,i}))\}_{0 \le i < n+s}.$$

4. Next, ME_1 publishes $\mathcal{L}^{(1)}$ on the bulletin board.

5. If the mixing party contains several mixing elements, ME_2, taking $\mathcal{L}^{(1)}$ as input, generates a random $n + s$ elements permutation π_2 and a random set $\{r'_{2,i}, r''_{2,i}\}_{0 \le i < n+s}$ and shuffles and re-encrypts the elements of $\mathcal{L}^{(1)}$ as described in step 3 generating $\mathcal{L}^{(2)}$ as output. The remaining mixing elements will operate in the same way until the last mixing element, ME_λ, produces as output the list $\mathcal{L}^{(\lambda)}$ containing $n+s$ encrypted votes (s of them are dummies).

Dummy tracing and removal

Once the last mixing element has published list $\mathcal{L}^{(\lambda)}$, tracing dummy votes from $\mathcal{L}$ to $\mathcal{L}^{(\lambda)}$ provides evidence that no mixing element has been bypassed during the shuffling procedure. In a first round, the mixing elements publish the part of their permutation that makes it possible to trace the path followed by dummy votes along the mixing process. After that, in the second round, the mixing elements progressively reveal the randomness employed for re-encrypting dummy votes. If all checks are properly satisfied, dummies will be finally removed. This procedure is formally described below:

1. ME_1 publishes the sorted lists

   $$L_0 = \{n, \ldots, n+s-1\}, \qquad L_1 = \{\pi_1(n), \ldots, \pi_1(n+s-1)\}$$

   and the commitment H_1 computed as

   $$H_1 = \mathcal{H}(r'_{1,n}|| \ldots ||r'_{1,n+s-1}||r''_{1,n}|| \ldots ||r''_{1,n+s-1}).$$

2. If the mixing party contains several mixing elements, the remaining ones will perform similarly, so that dummies can be traced up to $\mathcal{L}^{(\lambda)}$. More precisely, each mixing element ME_k publishes

   $$L_k = \{\pi_k(L_{k-1}(0)), \ldots, \pi_k(L_{k-1}(s-1))\}$$

   and the commitment H_k computed as

   $$H_k = \mathcal{H}(r'_{k,L_{k-1}(0)}|| \ldots ||r'_{k,L_{k-1}(s-1)}||r''_{k,L_{k-1}(0)}|| \ldots ||r''_{k,L_{k-1}(s-1)}).$$

3. The first mixing element ME_1 publishes $\{t_{1,j},\ t'_{1,j}\}_{0 \le j < s}$ so that everybody can generate the dummies

   $$\{(C_{n+j}, R_{n+j}) = (\mathrm{Enc}_y(1, t_{1,j}), \mathrm{Enc}_{y'}(1, t'_{1,j}))\}_{0 \le j < s}.$$

4. Sequentially, for k ranging from 1 to λ, the following steps are performed:

   (a) ME_k publishes

   $$\{r'_{k,L_{k-1}(0)}, \ldots, r'_{k,L_{k-1}(s-1)}, r''_{k,L_{k-1}(0)}, \ldots, r''_{k,L_{k-1}(s-1)}\}.$$

   (b) Everybody checks that hashing the elements in the previous list provides H_k as a result.

   (c) Everybody checks that

   $$C^{(k)}_{L_k(i)} = C^{(k-1)}_{L_{k-1}(i)} \cdot \mathrm{Enc}_y(1, r'_{k,L_{k-1}(i)}), \quad \text{for } 0 \le i < s,$$

   $$R^{(k)}_{L_k(i)} = R^{(k-1)}_{L_{k-1}(i)} \cdot \mathrm{Enc}_{y'}(1, r''_{k,L_{k-1}(i)}), \quad \text{for } 0 \le i < s.$$

5. If all checks have been properly satisfied, the s dummy votes of $\mathcal{L}^{(\lambda)}$ located at the positions in L_k are removed from $\mathcal{L}^{(\lambda)}$.

3.3.4 Tallying

When the last mixing element provides the list of votes without dummies, everybody can check the correctness of the mixing process. This is done by verifying the homomorphic aggregation of the ballots and ensuring that the redundancy in each ballot is still valid.

Homomorphic aggregation checking

First of all, everybody can check that the homomorphic aggregation of the received votes stored in list $\mathcal{L} = \{(C_i, R_i)\}_{0 \leq i < n}$ equals the aggregation of mixed votes in $\mathcal{L}^{(\lambda)} = \{(C_i^{(\lambda)}, R_i^{(\lambda)})\}_{0 \leq i < n}$. This is done as follows:

1. Everybody computes:

$$C_{total} = \prod_{i=0}^{n-1} C_i \quad \text{and} \quad R_{total} = \prod_{i=0}^{n-1} R_i$$

and also

$$C_{total}^{(\lambda)} = \prod_{i=0}^{n-1} C_i^{(\lambda)} \quad \text{and} \quad R_{total} = \prod_{i=0}^{n-1} R_i^{(\lambda)}.$$

2. The key storage trusted party verifiably decrypts the four ciphertexts so that everybody can check:

$$\mathrm{Dec}_x(C_{total}) = \mathrm{Dec}_x(C_{total}^{(\lambda)}) \quad \text{and} \quad \mathrm{Dec}_{x'}(R_{total}) = \mathrm{Dec}_{x'}(R_{total}^{(\lambda)}). \tag{3.1}$$

Redundancy checking

The previous checking is a necessary but not sufficient condition to ensure a proper mixing has been done. Sufficiency is provided by additionally checking that every mixed vote carries a correct redundancy. This redundancy checking has to be done in such a way that, in case of failure, no information about votes is leaked.

The next procedure checks that each vote $(C_i^{(\lambda)}, R_i^{(\lambda)})$ in $\mathcal{L}^{(\lambda)}$ satisfies

$$g'^{\mathrm{Dec}_x(C_i^{(\lambda)})} = \mathrm{Dec}_{x'}(R_i^{(\lambda)}).$$

This procedure requires some previous data to be generated before the election begins. Prior to the election, each mixing element ME_ℓ:

1. Generates $x_i^{(\ell)} \in G, r_i^{(\ell)} \in [1, p-1]$ at random and computes $P_i^{(\ell)} = \mathrm{Enc}_y(x_i^{(\ell)}, r_i^{(\ell)})$.

2. Publishes the commitment $Comm_i^{(\ell)} = (A_i^{(\ell)}, B_i^{(\ell)}) = (\mathcal{H}(P_i^{(\ell)}), g'^{x_i})$.

The previous step is repeated for $0 \leq i < n'$, where n' should be taken ensuring that $n \leq n'$ (n is the number of votes received).

Redundancy of mixed vote $(C_i^{(\lambda)}, R_i^{(\lambda)})$ is verified as follows:

1. Each mixing element ME_ℓ publishes $P_i^{(\ell)}$.
2. Everybody checks that $\mathcal{H}(P_i^{(\ell)}) = A_i^{(\ell)}$ for each $1 \leq \ell \leq \lambda$.
3. Everybody computes

$$C_i^{ch} = C_i^{(\lambda)} \prod_{\ell=1}^{\lambda} P_i^{(\ell)}.$$

4. The key storage trusted party verifiably decrypts $\mathrm{Dec}_x(C_i^{ch})$ and publishes the (pseudorandom)cleartext m_i' obtained.
5. ME_1 computes $R_i^{ch,1} = (R_i^{(\lambda)})^{x_i}$ and proves in zero-knowledge (Chaum-Pedersen's proof [CP93]) that $\log_{R_i^{(\lambda)}} R_i^{ch,1} = \log_{g'} B_i^{(1)}$.
6. The remaining mixing elements sequentially perform the same operation, taking as input the output of the previous mixing element, until ME_λ finally publishes $R_i^{ch} = \left(R_i^{ch,\lambda-1}\right)^{x_\lambda}$.
7. The key storage trusted party verifiably decrypts $\mathrm{Dec}_{x'}(R_i^{ch})$ and publishes the cleartext m_i'' obtained.
8. Everybody checks that $(g')^{m_i'} = m_i''$.

Vote decryption

If the checks in the previous steps are successful for each vote in $\mathcal{L}^{(\lambda)}$, the remaining stage is vote decryption. The mixed vote $(C_i^{(\lambda)}, R_i^{(\lambda)})$ is decrypted as follows:

1. Each mixing element ME_ℓ publishes $r_i^{(\ell)}$.
2. Everybody computes $x_i^{(\ell)} = \mathrm{Rev}_{r_i^{(\ell)}}(P_i^{(\ell)})$ and verifies that $g'^{x_i^{(\ell)}}$ equals $B_i^{(\ell)}$.
3. Everybody computes $m_i = m_i' / \prod_{\ell=1}^{\lambda} x_i^{(\ell)}$.
4. Finally, the submitted vote is obtained as $v_i = m_i \mathrm{div} 2^l$.

This is done sequentially for every mixed vote.

3.3.5 Failure recovery

Since participants prove the correct composition of the votes at vote reception, the only reason why some security checking may fail is due to mixing element cheating which would be a very rare case, since they are elements that are part of the voting platform.

In case some checking is not satisfied during dummy tracing (Section 3.3.3) or integrity checking (Section 3.3.4), every mixing element will be required to prove the correctness of its permutation using some backup classical zero-knowledge proof of mixing like [Nef01]. Those mixing elements not being able to execute a correct proof will be declared guilty. During vote decryption (Section 3.3.4), the checking of step 2 may fail because ME_ℓ has not provided a proper value for $r_i^{(\ell)}$. In this case, ME_ℓ is declared guilty.

After removing the guilty mixing elements, the mixing procedure can begin again. The important fact is that, in case of failure, no information about cleartext votes is revealed, so that another mixing operation can be performed taking the received votes as input and without the need to ask participants to perform any additional action.

3.3.6 Performance

In this section we analyze the cost of the computations that introduce delay between the end of the mixing and publication of the results (precomputable parts will not be addressed), focusing on the amount of exponentiations that is the most costly operation.

Being s the number of dummy votes, tracing them (Section 3.3.3) (undertaken after mixing) require the verifier to perform $4s$ plus $4s\lambda$ exponentiations for dummy generation (step 3) and tracing (step 4c), respectively. Mixing elements simply publish data with negligible cost. Homomorphic aggregation checking (Section 3.3.4) involves four verifiable decryptions with 12 and 8 exponentiations for the key storage trusted party and the verifiers, respectively.

The overall cost of redundancy checking and decryption of votes can be highly reduced by making use of batch verification techniques [BGR98]. Moreover, the cost of verifiable decryptions can be reduced by generalizing Chaum-Pedersen's proof [CP93] to prove the equality of n discrete logarithms in batch. Our analysis will take such optimizations into account.

The proof for redundancy requires the KSTP to perform $2n + 1$ exponentiations at step 4 plus the same amount at step 7. Steps 5 and 6 require each mixing element to perform n exponentiations with n (non aggregatable) Chaum-Pedersen proofs involving an overall amount of $4n$ exponentiations (mixing elements working in a pipeline permits them to work concurrently). On the side of the verifier, it requires $4(n + 1)$ exponentiations to verify steps 4 and 7 and λn verifiable decryptions involving $3\lambda n$ exponentiations.

Step 8 can be verified in $n/64$ exponentiations using [BGR98].

The cost of vote decryption is negligible for mixing elements (they only publish data) and requires $3\lambda n/64$ exponentiations for verification.

3.3.7 Security analysis

In this section, the system presented is formally proven to be secure in the sense of *integrity* and *privacy*.

For our system to be secure, the following assumptions should be met:

1. The *Key Storage Trusted Party* acts honestly by only decrypting the aggregated ciphertext.

2. The electoral roll has been properly generated (it only includes eligible voters).

3. Eligible voters' public keys have been properly certified by some certificate authority.

Assumptions 1 and 2 are required by any remote voting system in which public key certificates are used for authentication.

Under the previous assumptions, our proposal will now be proven to be secure.

3.3.7.1 Integrity

An e-voting protocol provides integrity if authentication, unicity and fairness properties are satisfied. The authentication property requires being able to differentiate between genuine voters and the rest. In our proposal, the ballots cast by voters are digitally signed and the public keys required to validate those signatures are correct (Assumption 2) and available in the electoral roll (Assumption 3). In this way, any entity can validate whether a given ballot comes from an eligible voter. Ballots without a valid signature are discarded.

Unicity requires the protocol to be able to detect any double voting attempt and prevent it. When a ballot is received, the ballot collection authority checks that the voter casting it has not voted before. Hence, just one ballot per voter is allowed.

Fairness

The following lemma and theorem prove the proposed voting system satisfies the *fairness* condition (no vote has been added, removed or modified during the mixing process).

Lemma 3.3.1. *Given* $\mathcal{L} = \{(C_i, R_i)\}_{0 \leq i < n}$ *with* $C_i = Enc_y(v_i)$ *and* $R_i = Enc_{y'}(g'^{v_i})$*, any entity aiming to generate a list*

$$\mathcal{L}^{(\lambda)} = \{(Enc_y(\hat{v}_i), Enc_{y'}(g'^{\hat{v}_i}))\}_{0 \leq i < n}$$

satisfying:

1. $\prod_{i=0}^{n-1} v_i = \prod_{i=0}^{n-1} \hat{v}_i \pmod{p}$,
2. $\sum_{i=0}^{n-1} v_i = \sum_{i=0}^{n-1} \hat{v}_i \pmod{p}$,
3. *No* n*-permutation* π *satisfies* $v_i = \hat{v}_{\pi(i)}$, *for* $0 \leq i < n$,

succeeds with probability at most 2^{-h}*, where* $h = \max_i\{H(v_i)\}$ *with* $H(v_i)$ *denoting the entropy on the knowledge of* v_i*.*

Proof. Trivial solutions where $\mathcal{L}^{(\lambda)}$ contains a permutation of the (possibly re-encrypted) elements in $\mathcal{L}$ are discarded by condition 3. Condition 1 permits us to model $\hat{v}_i = k_i v_i$, for $0 \leq i < n-1$ and $\hat{v}_{n-1} = (\prod_{i=0}^{n-2} k_i)^{-1} v_{n-1}$, for some values $k_0, \ldots, k_{n-2} \neq 0$ with at least one of them being different from 1.

If $n = 1$, conditions 1 and 2 require $k_0 = 1$ which violates condition 3. If $n = 2$, two possibilities exist, namely $k_0 = 1$ or $k_0 = v_1 v_0^{-1}$. The first case generates $\hat{v}_0 = v_0$ and $\hat{v}_1 = v_1$, while the second one leads to $\hat{v}_0 = v_1$ and $\hat{v}_1 = v_0$, both of them violating condition 3.

In the general case, $n > 2$, from condition 2 we obtain

$$\sum_{i=0}^{n-1} v_i = \sum_{i=0}^{n-2} k_i v_i + (\prod_{i=0}^{n-2} k_i)^{-1} v_{n-1},$$

which is equivalent to

$$\sum_{i=0}^{n-2} (1 - k_i) v_i = \left((\prod_{i=0}^{n-2} k_i)^{-1} - 1 \right) v_{n-1}. \tag{3.2}$$

The previous expression is a linear relation of values v_i where the constant coefficients depend on k_i. Since there exists a value v_j with h bits of entropy, once all values k_i have been chosen, v_j can be isolated from equation 3.2 so that the equality will hold with probability 2^{-h} at the most. If some entropy exists on the knowledge of other values v_i, the probability of success further decreases. □

Theorem 3.3.2. *If all checks performed in the voting system are satisfied, a corrupted mixing party that aims to modify the vote submitted by some honest participant will succeed with probability* 2^{-l} *at the most, where* l *is the parameter described in step 1 of the procedure in Section 3.3.2.*

Proof. The vote submitted by an honest participant contains l random bits, so that the corrupted mixing party has at least l bits of entropy about its knowledge. The checks of Formula 3.1 in Section 3.3.4 ensure conditions 1 and 2 of Lemma 3.3.1 are satisfied. Moreover, the redundancy checks and vote decryption phases ensure each tuple in $\mathcal{L}^{(\lambda)}$ has the form $\left(\mathrm{Enc}_y(\hat{v}_i), \mathrm{Enc}_{y'}\left(g'^{\hat{v}_i}\right)\right)$. Within these conditions, the upper bound on the probability of success follows from Lemma 3.3.1. □

Verifiability

The fairness and unicity requirements from the voting scheme can be checked in each of the steps of the election. For this reason, our proposal can be considered end-to-end verifiable. The unicity is first checked during the casting phase. Each ballot cast is digitally signed, allowing any entity to verify that its caster appears in the electoral roll and that she has not cast more than one ballot. Fairness can also be verified by checking whether the correctness proof of the shuffle holds. As demonstrated above, the probability of a cheating mixer modifying the ballots without being noticed during the mixing phase and later providing a valid proof of correct mixing is negligible. Finally, the ballots are verifiably decrypted, so the results obtained are the cleartext of the messages received by eligible voters who cast a ballot.

3.3.7.2 Privacy

The privacy property states that it must not be possible to link the content of any decrypted vote to the identity of the participant who cast it. Since votes are received in an authenticated manner, the mixing phase is needed so as to break the relation between the encrypted votes received and the ciphertexts that will be decrypted. In a mix-type e-voting system (properly using strong cryptography), privacy could be broken in three ways:

1. **Permutation disclosure**

   If the permutation applied to votes was revealed, it would be possible to relate each cleartext vote to the identity of the participant who cast it. When the mixing party is composed of several mixing elements, the overall permutation stays secret as long as at least one of the mixing elements does not reveal its individual permutation. The voting system must ensure all the mixing elements have taken part in the shuffling of the resulting set of mixed votes, *i.e.* no mixing element has been bypassed by a dishonest coalition. In our system, the addition and tracing of s dummy votes (Section 3.3.3) ensures that the probability of successfully bypassing an honest mixing element is $(s/n)^s$ at the most. The proof in [SMPP10] can be straightforwardly applied to our system.

2. **Message relation attacks**

   In a *relation attack* [Pfi95], a corrupted participant takes the encrypted vote submitted by some other participant, modifies it in some way, and then sends it as her own contribution. After vote decryption, the attacker will search for two related cleartexts that will permit her to know the contribution of the participant attacked. Some attacks over [GZB$^+$02] described in [Wik04] follow this approach. These attacks are not possible if participants are required to know the cleartext of the vote they submit. The following Lemma shows that this is the case in our system.

   Lemma 3.3.3. *A prover successfully completing the zero-knowledge proof of Section 3.3.2 on her submitted vote (C, R) knows the cleartext of C.*

   Proof. If the number of rounds is large enough, a successful proof ensures there exists some v so that $C = \text{Enc}_y(v)$ and $R = \text{Enc}_{y'}(g'^v)$. Also, a successful proof means that when the prover submitted her commitment, (C', R'), she was able to send a good response (z, d) whatever the value of the challenge *chall* was. Sending a proper response when $chall = 1$ ensures that $C' = \text{Enc}_y(vv')$ and $R' = \text{Enc}_{y'}(g'^{vv'})$ for some v'. Moreover, the value z sent by the prover can be used to obtain vv'. On the other side, when $chall = 0$, the value z can be used to obtain the cleartext of C'/C which is v'. Multiplying vv' by the inverse of v' gives v as a result, so that the prover has enough information to obtain v. □

3. **Vote marking**

   A vote marking attack is performed by the first mixing element (or a coalition including it) that incorrectly re-encrypts the vote cast by some participant so that, after vote decryption, her vote will be identified due to its corrupted state. If the voting system satisfies the integrity property this corruption will be detected, but the system must ensure that no confidential information has been leaked at this point. A corrupted coalition not including the first mixing element makes no sense since the set of votes they receive has already lost its link with the identity of participants, so that the colluders will be caught after having obtained no benefit. The following lemmas prove our system is secure against such attacks.

   Lemma 3.3.4. *Assuming there is at least one honest mixing element, the redundancy of the votes in $\mathcal{L}^{(\lambda)}$ is verified without revealing any information on their cleartext.*

Proof. The procedure for checking redundancy is described in Section 3.3.4. At step 4, ciphertext $C_i^{(\lambda)} \prod_{\ell=1}^{\lambda} P_i^{(\ell)}$ is decrypted, where each mixing element ME_ℓ has contributed with its own

$$P_i^{(\ell)} = \mathrm{Enc}_y\left(x_i^{(\ell)}\right).$$

The result of this decryption is $\mathrm{Dec}_x(C_i^{(\lambda)})x_i^{(1)} \cdots x_i^{(\lambda)}$. Next, at step 7, ciphertext $\left(R_i^{(\lambda)}\right)^{\hat{x}_i^{(1)} \cdots \hat{x}_i^{(\lambda)}}$ is decrypted obtaining value

$$\left(\mathrm{Dec}_{x'}(R_i^{(\lambda)})\right)^{\hat{x}_i^{(1)} \cdots \hat{x}_i^{(\lambda)}}.$$

Cleartexts $\mathrm{Dec}_x(C_i^{(\lambda)})$ and $\mathrm{Dec}_{x'}(R_i^{(\lambda)})$ remain unrevealed as long as at least one mixing elements keeps its values $x_i^{(\ell)}$ and $\hat{x}_i^{(\ell)}$ unrevealed. □

Note that the protocol states that $x_i^{(\ell)}$ must equal $\hat{x}_i^{(\ell)}$ but, this is not checked at this moment. This would permit the following attack in which the first mixing element, before the election begins, submits two invalid commitments (see Section 3.3.4):

$$Comm_i^{(1)} = (A_i^{(1)}, B_i^{(1)}) = (\mathcal{H}(P_i^{(1)}), g'^{x_i})$$

with $P_i^{(1)} = \mathrm{Enc}_y(k^{-1}x_i^{(1)}, r_i^{(1)})$ and

$$Comm_j^{(1)} = (A_j^{(1)}, B_j^{(1)}) = (\mathcal{H}(P_j^{(1)}), g'^{x_j})$$

with $P_j^{(1)} = \mathrm{Enc}_y(kx_j^{(1)}, r_j^{(1)})$.

After the votes have been submitted, ME_1 takes the input of two participants

$$(C_1, R_1) = (\mathrm{Enc}_y(v_1), \mathrm{Enc}_{y'}(g'^{v_1})) \text{ and } (C_2, R_2) = (\mathrm{Enc}_y(v_2), \mathrm{Enc}_{y'}(g'^{v_2}))$$

and modifies them so that, after mixing, C_1 and C_2 have been transformed into

$$C_1' = \mathrm{Enc}_y(kv_1) \text{ and } C_2' = \mathrm{Enc}_y(k^{-1}v_2),$$

respectively. This modification will satisfy the homomorphic aggregation checking (Section 3.3.4).

After mixing, if the mixed ciphertext corresponding to (C_1', R_1) occupies position i of $\mathcal{L}^{(\lambda)}$ and (C_2', R_2) is at the j-th position, which happens with probability $\approx 1/n^2$, the redundancy checking in the same Section will also be satisfied. Fortunately, the probability of success becomes very small for common values of n (number of participants) in any election. Moreover, if in spite of its reduced chance the previous attack succeeded, the corruption would be detected during decryption and the corrupted mixing element could be identified and punished.

Chapter 4

Homomorphic tallying paradigm

This chapter begins with Section 4.1, which is a brief description of the homomorphic tallying remote voting paradigm.

Section 4.2 presents the "array ballot" variant of the paradigm, in which ballots consist of an array of ciphertexts instead of a single one. After that, in Section 4.3, we present a novel contribution to the "array ballot" variant of the paradigm. More exactly, we propose a construction in which the correct composition of a ballot is proven by making use of a zero-knowledge proof of mixing. In this way, we avoid the use of the so-called *range proofs* and their elevated computational cost. The construction mentioned has been published in [MMS15].

4.1 Paradigm description

The homomorphic tallying paradigm requires the entities and is composed of the phases explained in Section 1.1. The only additional requirement of the paradigm is the use of a homomorphic public key cryptosystem. We next provide a detailed description of the procedures that take place in each phase:

1. **Setup:** A private key for the homomorphic cryptosystem is generated and distributed among the vote tallying authorities. After that, the election public key is published on the bulletin board. Next, the electoral roll is published including the identity and digital certificate of each eligible voter. In this paradigm, the list of candidates includes a value employed to represent each candidate. In the end, the bulletin board contains the election public key, the electoral roll and the candidate list.

2. **Vote casting:** During this phase, a voter can generate her ballot by selecting the value representing her choice and encrypting it under the election public key. Next, the voter generates a zero-knowledge proof which proves that her ballot properly represents a valid candidate choice. After that, the voter digitally signs her ballot using the private key associated with her digital certificate and sends the ballot, its zero-knowledge proof of correct composition and its signature to the vote collecting authority.

   Upon receiving a ballot, the vote collecting authority verifies that its caster appears in the electoral roll. This is done by checking the validity of the ballot signature using the digital certificate published on the bulletin board. If the caster is an eligible voter and she has not cast a ballot before, then the vote collecting authority proceeds with the verification of the zero-knowledge proof. If the checks performed hold, the vote collecting authority publishes all the data received on the bulletin board for public verifiability.

   The vote casting phase ends when the voting period expires or when all the eligible voters have cast their ballots.

3. **Tallying:** Once the vote casting phase has ended, the vote tallying authorities compute the ciphertext resulting from the homomorphic aggregation of all the ballots. The tallying authorities verifiably decrypt this ciphertext, gather the result of the election and publish it on the bulletin board.

Figure 4.1 depicts a set of voters $v_1, \ldots, v_n$. Each voter v_i casts a ballot C_{v_i} which is an encryption of her cleartext vote M_i. After that, the ballot collection authority aggregates all the ballots received obtaining ciphertext T as a result. By decrypting T, we obtain $M_1 + \cdots + M_n$ (an additive homomorphic cryptosystem is assumed).

Notice that, if a vote for candidate k was represented by a ciphertext $C = \text{Enc}(P_k)$, a malicious voter could send a ciphertext $C' = \text{Enc}(9P_k)$ representing 9 votes for candidate k. This would violate the unicity requirement of the election and would invalidate the paradigm. To prevent this, all the ballots are cast together with a zero-knowledge proof, ensuring the cleartext of each ballot belongs to a set of valid values.

An approach to construct proofs of partial knowledge was presented by Cramer et al. in [CDS94]. This approach has been implemented using range proofs, which demonstrate that the cleartext encrypted in a ciphertext lies in a known set. Unfortunately, for electronic voting purposes these proofs are not efficient enough, since their cost increases linearly with the number of elements in the set and, in some elections, this set can be really large. Recently, more efficient proofs have been proposed for specific cryptosystems

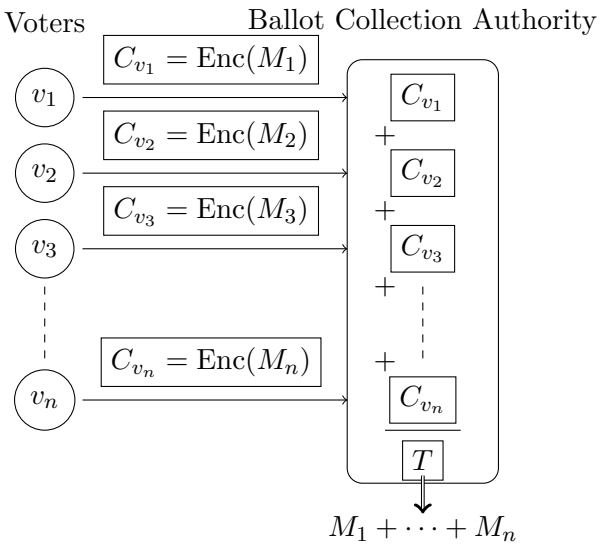

Figure 4.1: Additive homomorphic tallying remote voting.

like that detailed in [PB09] for the Paillier cryptosystem [Pai99]. Nevertheless, it also restricts the configuration of the messages and requires several rounds of interaction between the prover and the verifier.

Homomorphic tallying proposals

Electronic voting schemes in this paradigm either use additive or multiplicative homomorphisms. Some drawbacks exist in both cases.

Several e-voting proposals [CLW08, DJ01, Gro05] implementing the homomorphic tallying paradigm use the Paillier cryptosystem due to its additive homomorphic property. Most of the proposals using an additive homomorphic cryptosystem generate the set of values representing each candidate by following a simple pattern: if the election has n eligible voters, and the amount of candidates is k, the set is as follows:

$$\{n^0, n^1, \ldots, n^{k-1}\}.$$

If the candidate list was $\{L_1, \ldots, L_k\}$, a voter casting a ballot for a candidate L_l would send an encryption of n^{l-1}, together with a range proof demonstrating that the cleartext of its ballot falls in that set. When the election ends, the ciphertexts are homomorphically aggregated into a single ciphertext. Any entity can verify that such aggregation has been performed properly by computing it by itself. After that, the aggregated ciphertext is

decrypted, resulting in a cleartext of the form

$$\sum_{i=1}^{k} x_i \cdot n^{i-1},$$

in which x_i is the amount of votes for candidate L_i. Notice that the set of possible messages has been chosen in a way that, after homomorphically aggregating all the votes, the decrypted value represents the election result in a non-ambiguous way (vector $(x_1, \ldots, x_k)$ can be obtained).

The implementation of this paradigm with an additive homomorphic cryptosystem offers an efficient tallying phase requiring only one decryption and an easy and fast way to obtain the election result from the cleartext of the homomorphically aggregated ciphertext.

Efficiency is a very important requirement for e-voting systems, but it is not the only one. For security, the private key of the election should be distributed among the tallying authorities. Unfortunately, most of additive homomorphic cryptosystems use the factorization problem as a trapdoor, so their distributed key generation algorithms have an elevated cost.

On the other hand, there are several electronic voting proposals [PAB+04, PB11] implementing the homomorphic tallying paradigm using ElGamal cryptosystem which offers a multiplicative homomorphic property. In such proposals, a common solution is to generate the set of values representing each candidate as a set containing k prime numbers:

$$\{s_1, s_2, \ldots, s_k\},$$

being k the number of candidates. Hence, the cleartext obtained after decrypting the aggregated ciphertext is of the form

$$\prod_{i=1}^{k} s_i^{x_i},$$

with x_i representing the amount of votes for candidate L_i. With these proposals, the ciphertexts have to be homomorphically operated in groups. This measure is required to prevent a cleartext range overflow which would prevent the election result from being obtained. Hence, instead of a single aggregated ciphertext, these proposals generate several. Each of them will be individually decrypted and tallied.

As a result, the amount of verifiable decryptions increases so that its cost moves from an $O(1)$ cost in additive schemes to $O(n)$ for multiplicative ones, where n is the number of voters. Despite ElGamal providing an efficient distributed key generation algorithm, the resulting solutions present a lack of performance when the amount of candidates increases. This is because this fact causes a reduction in the amount of ballots per group, so that more aggregated ciphertexts need to be generated and decrypted.

Multiplicative homomorphic cryptosystems can be adapted to become an additive homomorphism. Several proposals, like [HS00, LK03], are based on the additive modification of ElGamal cryptosystem. Due to the additive homomorphic property of the cryptosystem, the set of possible values chosen for these proposals is:

$$\{n^0, n^1, \ldots, n^{k-1}\}.$$

As occurs with additive schemes, the ballot aggregation at the end of the voting phase generates just one aggregated ciphertext. In the additive modification of ElGamal cryptosystem, the cleartext obtained after decrypting the aggregated ciphertext is of the form:

$$g^{\sum_{i=0}^{k-1} x_i \cdot n^i}.$$

Thus, to obtain the values x_i, the ballot collection authority has to solve a discrete logarithm problem, which is hard to compute when the solution range increases. This space range grows exponentially with the amount of candidates k.

In conclusion, it can be stated that, despite the efforts to provide an efficient homomorphic tallying e-voting scheme, the different approaches, either additive or multiplicative, require some improvements permitting their use in large-scale elections.

4.2 Array ballots

With this variant [AMPQ09], a ballot consists of a ciphertext array in which each element represents the eventual vote for one of the candidates:

$$(C_1, C_2, \ldots, C_k)\,.$$

Each of the k components of the ballot represents a different candidate. A voter voting for candidate j generates her ballot so that C_j is an encryption of a specific point V while the remaining ciphertexts are an encryption of $\mathcal{O}$. By component-wisely aggregating all the ballots received, we obtain a vector whose j-th component is an encryption of $n_j V$, being n_j the amount of votes for candidate j. Given V and $n_j V$, finding n_j requires solving a discrete logarithm problem which is easy to compute when n_j is small or known to fall in a given small range.

A standard e-voting scheme using the array ballot configuration would be as follows.

4.2.1 Participating parties

The protocol involves the following parties:

The following description assumes the elliptic ElGamal cryptosystem is being used.

- *Electoral roll authority:* Publishes the electoral roll on the BB. Let n be the amount of potential voters in the electoral roll.

- *Key Storage Trusted Party (KSTP):* Generates and stores a private key and publishes the election public key on the BB. It will perform a verifiable decryption when required. It may be a distributed entity.

- *Voters:* Select a candidate and send an encrypted ballot together with a zero-knowledge proof proving its correct composition.

- *Ballot Collection Authority (BCA):* Collects the ballots and verifies their correctness. It also checks the identity of their caster and verifies that she appears in the electoral roll. If all the checks are satisfied, the ballot received is published on the bulletin board so that any external entity can check its validity.

We assume there exists a publicly readable bulletin board (BB). Everyone can access the information on the BB but only the electoral roll authority, the KSTP, and the BCA can write on it.

4.2.2 Protocol

The protocol is divided into three phases. Each one starts when the previous one ends. The following description assumes an election in which a single candidate is voted in each ballot. A notation table is given in Table 4.1.

n	Number of voters
k	Number of candidates
v_i	i-th voter
L_j	j-th candidate
C_{v_i}	Ballot generated by voter v_i
$C_{i,j}$	EC-ElGamal ciphertext located at the j-th position of v_i's ballot
$\mathcal{O}$	Elliptic curve point at infinity
V	An elliptic curve point
Q	Election public key
T	Array of aggregated ciphertexts
T_j	EC-ElGamal ciphertext located at the j-th position of T
R_j	Cleartext of T_j

Table 4.1: Notation table.

Setup phase

In this phase, the system is configured and some information is published. First of all, the electoral roll authority publishes on the BB:

- The electoral roll with all the voters $\{v_1, v_2, \ldots, v_n\}$ authorized to cast a ballot and their public keys $pk_1, pk_2, \ldots, pk_n$ which could be certified by some certificate authority.
- A sorted list of candidates $L = \{L_1, L_2, \ldots, L_k\}$.

The $KSTP$ chooses a secret key d and publishes a point P of prime order m of an elliptic curve $E(\mathbb{F}_p)$ together with the public key $Q = dP$ on the bulletin board. To improve the reliability on the $KSTP$, it can be distributed into a set of entities $\{KSTP_1, \ldots, KSTP_t\}$. Then, each $KSTP_i$ generates its own private key d_i and publishes $Q_i = d_iP$. The election public key is $Q = \sum_i Q_i$. A message encrypted under Q can only be decrypted if all the entities composing the $KSTP$ collaborate.

The BCA publishes a point V of the same curve. Then, it pre-computes and stores the points $\{V, 2V, \ldots, (n/2)V\}$. Each point xV is stored together with the corresponding integer x.

Vote casting phase

When the vote casting phase starts, each voter v_i may cast her ballot for one candidate, taking into account the candidate order published on the bulletin board. The ciphered ballot consists of an array of ballots $(C_{i,1}, \ldots, C_{i,k}, C_{i,k+1})$, where each $C_{i,j}$ is related with the candidate L_j. Ciphertext $C_{i,k+1}$ refers to the blank vote. In order to generate each ciphertext, the voter randomly generates $r_{i,1}, r_{i,2}, ..., r_{i,k}, r_{i,k+1} \in_R \{1, ..., m-1\}$.

After that, v_i generates k ciphertexts with an elliptic ElGamal encryption of V for the non-chosen candidates, and an additional ciphertext with the encryption of $2V$ for the chosen candidate. We assume, for simplicity, that the chosen candidate is the last one, k, so, in the end, v_i generates:

$$C_{i,j} = (r_{i,j}P, V + r_{i,j}Q), \quad \text{for } \ j \neq k,$$

and

$$C_{i,k} = (r_{i,k}P, 2V + r_{i,k}Q).$$

After that, v_i signs her ballot, which is a message containing all her ciphertexts, and generates a zero-knowledge proof of correctness for her ballot, ensuring that her ballot has been coded properly, using the procedure described in Section 4.2.3. After that, she sends $(C_{i,1}, C_{i,2}, ..., C_{i,k}, C_{i,k+1})$, her signature and the zero-knowledge proof to the BCA.

When the BCA receives a ballot, it proceeds as follows:

- Verify the signature using the voter's public key which is available on the BB.
- Check that the voter has not voted before.
- Check that there is no other ciphertext with the same points of $C_{i,j}$ for each $j \in \{1, \ldots, k+1\}$.
- Verify the zero-knowledge proof of correct ballot composition.
- If all the checks are satisfied, publish the ballot, its zero-knowledge proof and its signature on the BB. Otherwise, the ballot is discarded.

Tallying phase

When the vote casting phase has ended, the BCA aggregates the ballots received by computing, for each $j \in \{1, \ldots, k+1\}$:

$$T_j = \sum_{i=1}^{z} C_{i,j} = \left(\sum_{i=1}^{z} A_{i,j}, \sum_{i=1}^{z} B_{i,j} \right),$$

where z is the amount of voters that have cast a vote. Then, the BCA asks the KSTP to perform a verifiable decryption of each ciphertext T_j, $j \in \{1, \ldots, k+1\}$, obtaining

$$Dec(T_j) = \sum_{i=1}^{z} y_{i,j} V, \quad y_{i,j} \in \{1, 2\}.$$

The range of possible values for $\sum_{i=1}^{z} y_{i,j}$ is $\{z, z+1, \ldots, 2z\}$. Therefore, the ballot collection authority computes $Dec(T_j) - zV = x_j V$ and then searches for $x_j V$ in the pre-computed table generated during the setup phase, and recovers x_j. If $x_j V$ is not in the table, it means that the candidate L_j has received more than $n/2$ votes. In this case the BCA computes $x'_j V = x_j V - n/2V$, which is guaranteed to have been pre-calculated and obtains x'_j. The amount of votes for candidate j is $x_j = n/2 + x'_j$.

Value T_{k+1} represents the amount of blank votes.

4.2.3 Zero-knowledge proof

During the vote casting phase, a voter must prove in zero-knowledge the correctness of her ballot. For the standard protocol, a non-interactive proof presented in [CDS94], also used for Helios 2.0 in [AMPQ09], is adapted for being used with elliptic curve ElGamal ciphertexts.

Helios 2.0 used the same proof applied to the aggregation of the components of each ballot in a way that is only suitable for proving that the voter has voted for only one candidate.

The proof presented in this section takes advantage of the reveal operation. It permits us to prove efficiently that a voter cannot vote for more candidates than allowed (in a multiple choice election, more than one candidate can be chosen).

The main idea is that the prover (voter) convinces the verifier (BCA or anyone) that each of the components of her ballot is an encryption of either V or $\mathcal{O}$. Next, she proves that only one of them is encrypting V by revealing the value of their aggregation. This avoids the need to use a range proof.

Prover

First of all, the prover (voter v_i) has to prove in zero-knowledge that each ciphertext $C_{i,j}$ in vector

$$C_{i,1} = (A_{i,1}, B_{i,1}), \ldots, C_{i,k+1} = (A_{i,k+1}, B_{i,k+1})$$

is an encryption of either V or $\mathcal{O}$. In order to do that, the prover makes the following computations. For each j, if $C_{i,j}$ is an encryption of V, the prover proceeds as follows:

1. Randomly generate $w''_j, u''_j, s_j \in_R \{1, \ldots, m-1\}$.

2. Compute,

$$\begin{array}{ll} A'_j = s_j P, & B'_j = s_j Q, \\ A''_j = w''_j P + u''_j A_{i,j}, & B''_j = w''_j Q + u''_j (B_{i,j} - \mathcal{O}), \end{array}$$

3. Compute,

$$\begin{array}{l} chall_j = \mathcal{H}(A'_j, A''_j, B'_j, B''_j), \\ u'_j = chall_j - u''_j, \\ w'_j = s_j - u'_j r_{i,j}, \end{array}$$

   where $\mathcal{H}$ is a cryptographic hash function like SHA256 [NIS94]. Notice that $r_{i,j}$ is the random integer taken in the generation of $C_{i,j}$.

If $C_{i,j}$ is an encryption of $\mathcal{O}$, the prover will generate A''_j, B''_j as A'_j, B'_j and vice versa, taking into account that the computation of B'_j will involve V instead of $\mathcal{O}$. The generation of u'_j and w'_j will also be swapped with u''_j and w''_j, respectively.

After that, the prover computes $r_i = r_{i,1} + r_{i,2} + \cdots + r_{i,k+1} \pmod m$ and sends

$$A'_j, A''_j, B'_j, B''_j, u'_j, u''_j, w'_j, w''_j$$

for each j, $1 \leq j \leq k+1$, together with r_i to the verifier.

Verifier

For each j, $1 \leq j \leq k+1$, the verifier checks that

$$\begin{array}{ll} A'_j = w'_j P + u'_j A_{i,j}, & B'_j = w'_j Q + u'_j (B_{i,j} - V), \\ A''_j = w''_j P + u''_j A_{i,j}, & B''_j = w''_j Q + u''_j (B_{i,j} - \mathcal{O}), \\ \mathcal{H}(A'_j, A''_j, B'_j, B''_j) = u'_j + u''_j. & \end{array} \tag{4.1}$$

After that, the verifier aggregates $C_i = C_{i,1} + C_{i,2} + \cdots + C_{i,k+1}$ and uses r_i to reveal $C_i = (A_i, B_i)$, as shown in Section 2.1.1, and checks that

$$\mathrm{Rev}_{r_i}(C_i) = B_i - r_i Q = V.$$

All these checks ensure that v_i has voted for only one candidate.

4.2.4 Efficiency

The protocol requires the voters to generate k ballots and the corresponding proofs. Later on, the BCA verifies the proofs, aggregates the ballots and performs k decryptions. In this way, the most consuming process is the generation and verification of the proof of correctness. For instance, in an election in which a single candidate has to be chosen, it has to be proven that each component of the ballot cast is an encryption of either V or $\mathcal{O}$, together with an additional proof that just one of the components is an encryption of V.

This last part can be proven by showing that the homomorphic aggregation of all the components of the vector is an encryption of V. In an election where each voter can choose between 0 and ℓ_{max} candidates, the aggregation of all the vector components has to be proven to be an encryption of an element in $\{\mathcal{O}, V, 2V, \ldots, \ell_{max}V\}$. Computing and verifying a range proof is expensive in cost, specially when the range of valid cleartexts is large. For instance, in the adaptation in Section 4.2.3, generating and verifying a proof showing that the cleartext of a given ciphertext belongs to a set with $(\ell_{max}+1)$ elements requires us to compute $3(\ell_{max}+1)$ and $4(\ell_{max}+1)$ exponentiations, respectively.

Considering an election in which we can vote for up to ℓ_{max} out of k candidates, the amount of exponentiations needed for a voter to generate the k ballot components is $2k$ (this is the cost of composing k ElGamal ciphertexts). After that, proving that each ballot component is an encryption of $\mathcal{O}$ or V requires the computation of $6k$ exponentiations (each binary range proof involves 6 exponentiations). Finally, proving that no more than ℓ_{max} candidates have been chosen has an additional cost of $3(\ell_{max}+1)$ exponentiations.

Verifying the previous proofs requires the computation of $8k+4(\ell_{max}+1)$ exponentiations per ballot. Hence, in an election with n participants, the amount of exponentiations to be computed by the ballot collection authority (or an external entity) to verify all the ballots is $(8k + 4(\ell_{max}+1))n$.

4.3 Mixings on array ballots

A new hybrid proposal for remote voting was presented in [MMS15]. The proposal works in the same manner as homomorphic tallying systems. Ballots consist of a ciphertext array, but the correctness of proper ballot coding is not proven by means of range proofs. Instead, proper ballot coding is proven using a zero-knowledge proof of mixing.

In this way, the proposal offers all the advantages of the homomorphic tallying paradigm, while it avoids the elevated computational cost of range proofs. The resulting system takes advantage of the reduced cost of recent proposals for proving the correctness of a mixing operation.

The proposal is universally verifiable and can accommodate elections where each voter can vote for more than one candidate (multiple-candidate elections), as shown in Section 4.3.3. The description given below assumes an election in which voters choose one candidate out of k.

4.3.1 Participating parties

The parties involved in the protocol are exactly the same as those required for the standard protocol presented in Section 4.2. We assume that there is a publicly readable bulletin board on which data such as the ballots received are published, so that external entities can verify the correctness of the process. The following parties are involved in the election: *electoral roll authority, Key Storage Trusted Party (KSTP), voters and the Ballot Collection Authority (BCA).*

The bulletin board is publicly accessible for reading but only the electoral roll authority, the KSTP and the BCA can write on it.

4.3.2 System description

The protocol is composed of three stages, which are explained below.

4.3.2.1 Setup stage

In this preliminary stage, the system is configured: the electoral roll, the list of candidates and the required cryptographic keys are generated and published.

The electoral roll authority publishes (on the bulletin board):

- The electoral roll containing all the eligible voters $\{v_1, \ldots, v_n\}$ with their public keys $\{pk_1, \ldots, pk_n\}$, which could be certified by some certificate authority.
- A sorted list of candidates $L = \{L_1, \ldots, L_k\}$.

Meanwhile, the $KSTP$:

- Chooses and publishes an elliptic curve E defined over $\mathbb{F}_p$, whose cardinality is divisible by a 256 bits prime number m together with an order m point $P \in E(\mathbb{F}_p)$.
- Generates a private key d and publishes the public key $Q = dP$.

The KSTP stores the private key d in a safe place. To provide better security, the KSTP can be a distributed set of entities $\{KSTP_1, \ldots, KSTP_t\}$. In this case, each $KSTP_i$ generates its own private key d_i and publishes $Q_i = d_i P$. The election public key is $Q = \sum_i Q_i$. A message encrypted under Q can only be decrypted if all these entities collaborate.

Next, the BCA:

- Publishes a point $V \in E(\mathbb{F}_p)$ generated as $V = rP$ for a random r. After computing V, the value r can be discarded.
- Assuming an election with k candidates, generates a vector of ciphertexts $C = (C_1, C_2, \ldots, C_k)$ computed as:
  - Generates r_1 at random and computes:

$$C_1 = (r_1 P, V + r_1 Q),$$

  - For $j = 2, \ldots, k$: generates a random integer r_j and computes:

$$C_j = (r_j P, r_j Q).$$

  The vector C and the values r_j, for $1 \leqslant j \leqslant k$ are published.

Next, the BCA pre-computes the points $\{V, 2V, \ldots, nV\}$ and stores them. Each point aV is stored together with the corresponding integer a.

4.3.2.2 Vote casting stage

During this stage, the BCA is open for ballot reception. Each voter v_i may cast her ballot for one candidate. The ballot consists of a vector of ciphertexts $(C_{i,1}, \ldots, C_{i,k})$, in which $C_{i,s}$ is an encryption of V if v_i is voting for candidate L_s; otherwise, $C_{i,j}$ is an encryption of $\mathcal{O}$ for $j \neq s$. The voter v_i proceeds as follows:

1. Verify that the vector $C = (C_1, \ldots, C_k)$ published on the bulletin board satisfies $\text{Rev}_{r_j}(C_j)$ returns V for $j = 1$, and $\mathcal{O}$ for $1 < j \leq k$.
2. Generate at random a set of integers $\{r_{i,j}\}_{1 \leq j \leq k}$ such that $1 \leq r_{i,j} \leq m - 1$.
3. Create a permutation π satisfying $\pi(s) = 1$.

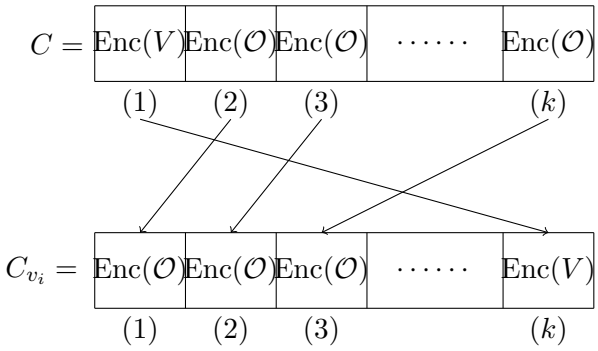

Figure 4.2: Ballot generation in an election with k candidates.

4. Permute C into $C'_{v_i} = (C'_{i,1}, \ldots, C'_{i,k})$ such that $C'_{i,j} = C_{\pi(j)}$.

5. Re-encrypt each $C'_{i,j}$ into $C_{i,j}$ using $r_{i,j}$ as re-encryption factor. Let $C_{v_i} = (C_{i,1}, \ldots, C_{i,k})$ be the resulting vector.

6. Generate a zero-knowledge proof showing that C_{v_i} is the result of applying a mixing operation to C. Let ZKP_{v_i} be the data generated as a result.

After that, v_i generates a message composed of vector C_{v_i} and ZKP_{v_i}. This message is digitally signed and sent to the BCA.

When the BCA receives a ballot from voter v_i, it proceeds as follows:

1. Check that v_i appears on the electoral roll and she has not voted before. Then, verify the digital signature of the ballot.

2. Check that the vector C_{v_i} contained on the ballot does not equal the vector of any previously received ballot.

3. Verify the zero-knowledge proof of mixing ZKP_{v_i}.

4. If all the checks are satisfied, publish the ballot on the bulletin board. Otherwise, the ballot is discarded. All these checks can also be performed by any external entity.

Figure 4.2 shows how vector C would be in an election with k candidates. A ballot for candidate L_k would be generated by permuting (and re-encrypting) the components of C so that the k-th one is an encryption of V.

4.3.2.3 Tallying stage

When the vote casting stage concludes, the BCA has the vectors received $\{C_{v_1}, \ldots, C_{v_z}\}$ (for simplicity we assume that the voters who participated are $v_1, \ldots, v_z$). The BCA computes an array of homomorphically aggregated ciphertexts $T = (T_1, \ldots, T_k)$ where each T_j, $1 \leq j \leq k$, is computed as:

$$T_j = \sum_{i=1}^{z} C_{i,j}.$$

Then, the BCA asks the KSTP to perform a verifiable decryption of each ciphertext T_j composing T. That is,

$$R_j = Dec(T_j).$$

Each point R_j obtained is of the form $n_j V$, for some integer n_j in the set $\{0, \ldots, z\}$ (if $n_j = 0$, then R_j is $\mathcal{O}$). Then, the ballot collection authority searches for R_j in the pre-computed table generated during the setup phase, and publishes n_j as the amount of votes for candidate L_j.

An external entity can check that vector T was properly computed by performing the computations itself. Next, it can check that the decryption of each component T_j really generates R_j as output (a verifiable decryption is performed) and verify that $n_j V$ equals R_j, for each j.

Figure 4.3 depicts how the ballot collection authority receives the ballots of the form $C_{v_i} = (C_{i,1}, \ldots, C_{i,k})$ which are then aggregated into an array $(T_1, \ldots, T_k)$, whose components will be decrypted at the end of the voting period.

4.3.3 Multi-candidate elections

The scheme presented above is able to run elections in which voters can select several candidates. We describe two cases:

1. Each voter has to select a fixed amount ℓ out of k candidates. In such a case, the vector $C = (C_1, \ldots, C_k)$ is generated as:

   $$C_j = (r_j P, V + r_j Q), \quad \text{for } 1 \leq j \leq \ell,$$

   and

   $$C_j = (r_j P, r_j Q), \quad \text{for } \ell < j \leq k.$$

   A voter voting for candidates $L_{s_1}, \ldots, L_{s_\ell}$ generates a permutation π satisfying $\pi(s_j) = j$ for $1 \leq j \leq \ell$.

2. Voters can choose between 0 and ℓ_{max} candidates. In such a case, vector C has $k + \ell_{max}$ components. They are generated as:

   $$C_j = (r_j P, r_j Q), \quad \text{for } 1 \leq j \leq k,$$

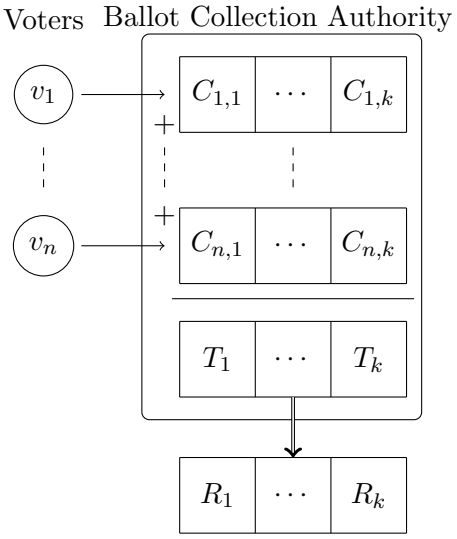

Figure 4.3: Ballot aggregation and decryption.

and

$$C_j = (r_j P, V + r_j Q), \quad \text{for } k+1 \leq j \leq k+\ell_{max}.$$

A voter voting for ℓ candidates $L_{s_1}, \ldots, L_{s_\ell}$ generates a permutation π satisfying $\pi(s_j) = k + j$ for $1 \leq j \leq \ell$ and $\pi(j) = j$ for $k + \ell + 1 \leq j \leq k + \ell_{max}$. If 0 candidates are selected, she takes π to be the identity permutation. At vote tallying, only the first k components of each vector will be aggregated.

4.3.4 Performance

In this section, we analyze the amount of exponentiations to be computed to generate and validate a ballot. We will consider a multi-candidate election in which each voter can choose between 0 and ℓ_{max} out of k candidates. The cost for single-candidate or multi-candidate with a fixed amount of election candidates selected is lower.

4.3.4.1 Ballot generation

First of all, a voter has to check that vector C was properly generated. This involves $2(k + \ell_{max})$ exponentiations (two per component). After that, the ballot components have to be shuffled and re-encrypted. Re-encrypting each component involves 2 exponentiations, hence $2(k + \ell_{max})$ additional exponentiations are computed. Finally, the cost of generating a zero-knowledge proof of correct mixing requires the computation of $5(k+\ell_{max})$ using [GL07]

or $4(k+\ell_{max})$ using [Pen11a]. Hence, a ballot can be generated by computing $8(k+\ell_{max})$ exponentiations.

As shown in Section 4.2, generating a ballot for an equivalent election using range proofs would require the computation of $8k+3(\ell_{max}+1)$ exponentiations.

Hence, the cost of generating a ballot in our proposal is slightly higher than using range proofs. Nevertheless, each voter just needs to generate one ballot per election so the cost of this step is not relevant.

4.3.4.2 Ballot correctness verification

With our proposal, verifying the correct composition of a ballot requires the verification of a proof of correct mixing of $k+\ell_{max}$ ciphertexts. This requires the computation of $4(k+\ell_{max})$ or $2(k+\ell_{max})$ exponentiations using [GL07] or [Pen11a], respectively. Hence, the ballots in an election with n eligible voters can be verified by computing $2(k+\ell_{max})n$ exponentiations (assuming all the voters have participated).

As shown in Section 4.2, verifying the ballots cast in an equivalent election using range proofs would require the computation of $(8k+4(\ell_{max}+1))n$ exponentiations.

Hence, our proposal provides a more efficient (four times faster if $k \gg \ell_{max}$) way to verify the correctness of the ballots cast. Furthermore, if a new zero-knowledge proof of mixing provided a better performance, then our proposal would also benefit from that.

4.3.4.3 Overall performance

If $k \gg \ell_{max}$, the amount of exponentiations required to verify all the ballots collected can be simplified to $O(kn)$. The cost of homomorphically aggregating n ballots is $O(kn)$, the cost of decrypting the aggregated ballot (it is composed of k ciphertexts) is $O(k)$ and the cost of decoding the cleartexts obtained is also $O(k)$.

Hence, the overall cost of our system is $O(kn)$. Since the amount of voters is usually much larger than the amount of candidates, that is $n \gg k$, we can conclude that our system has a cost which is linear with the number of voters.

Note that, with any remote voting system, the ballot collection authority receives an $O(n)$ amount of ballots that have to be authenticated, usually by means of digital signature checking. The number of signatures to be checked is $O(n)$, which turns out to be a lower bound on the performance of such a system.

4.3.5 Security analysis

A secure electronic voting system has to fulfill the security requirements.

For our system to be secure, the following assumptions should be met:

1. The *Key Storage Trusted Party* acts honestly by only decrypting the aggregated ciphertext.
2. The electoral roll has been properly generated (it only includes eligible voters).
3. Eligible voters' public keys have been properly certified by some certificate authority.

Note that Assumption 3 is required by any voting system belonging to the homomorphic tallying paradigm. Assumptions 1 and 2 are required by any remote voting system in which public key certificates are used for authentication.

Under the previous assumptions, our proposal will now be proven to be secure.

4.3.5.1 Integrity

An e-voting protocol provides integrity if authentication, unicity and fairness properties are satisfied. The authentication property requires being able to differentiate between genuine voters and the rest. The ballots cast by voters are digitally signed and the public keys required to validate those signatures are correct (Assumption 3) and available in the electoral roll (Assumption 2). In this way, any entity can validate whether a given ballot comes from an eligible voter. Ballots without a valid signature are discarded.

Unicity

When a ballot is received, the ballot collection authority checks that the voter casting it has not voted before. Hence, just one ballot per voter is allowed.

In homomorphic tallying remote voting, the previous check is not enough to ensure the unicity property. This property also requires that a malicious voter is not able to cast an improperly generated ballot including several votes for some candidate (if some component included, for instance, an encryption of $2V$ or $3V$) or votes for more candidates than allowed. This is the reason why this paradigm requires a proof showing that cast ballots have been properly generated.

With our proposal, this is proven by showing that a ballot C_{v_i} was obtained after mixing the components of the vector C published by the ballot collection authority during the setup stage. The next lemma addresses this issue.

Lemma 4.3.1. *If the proof of mixing provides* correctness*, our proposal provides* unicity*.*

Proof. Let us assume that our proposal does not provide unicity. That is, an adversary $\mathcal{A}$ can run two algorithms Alg_1, Alg_2 that generate a valid ballot which is not the result of mixing the ciphertexts of vector C. More formally, $Alg_1(C) \rightarrow C_{v_i}$ where C_{v_i} is a vector whose component cleartexts do not equal those of C, and $Alg_2(C, C_{v_i}) \rightarrow ZKP_i$ where ZKP_i is a valid zero-knowledge proof of mixing between vectors C and C_{v_i}.

Under that assumption, let us consider a mix-type voting system employing the previous proof of correct ballot mixing. Let C be the set of ballots cast by the voters. A misbehaving mixer could change the votes of an election as follows: The mixer would use $Alg_1(C) \rightarrow C'$, generating a set of ballots C' that are not a mixing of C. After that, the misbehaving mixer would use Alg_2 to generate a valid zero-knowledge proof of the mixing by running $Alg_2(C, C') \rightarrow ZKP$. Any entity verifying ZKP would be convinced that C' is a correct mixing of C, which is not the case. Hence the proof of mixing would not provide correctness and the claim follows. □

Fairness

Ensuring that partial results cannot be known depends on the privacy of the system and the correct behaviour of the KSTP (Assumption 1). From the already proven privacy property, we know that no information about the votes can be obtained from the cast ballots. Hence, fairness is provided as long as the KSTP behaves correctly and only decrypts the aggregated ballot at the end of the election. For a greater security, the KSTP can be a distributed entity as explained in Section 4.3.2.1.

Verifiability

The correctness of the whole voting process can be checked. Our proposal offers end-to-end verifiability. The votes cast are digitally signed so that any entity can verify that they come from people in the electoral roll and that no more than one ballot per eligible voter has been received. The proof of correct ballot composition and the ballot aggregation step can also be verified. Finally, decryption of the aggregated ballot is performed verifiably.

4.3.5.2 Privacy

The privacy property states that it must not be possible to link the content of any decrypted vote with the identity of the participant who cast it. In homomorphic tallying remote voting, only the aggregated ciphertext is decrypted (Assumption 1). In this way, just the global vote tally is obtained. In our proposal, the ballots are generated by mixing a publicly known vector of ciphertexts. The following lemma shows that the privacy of our proposal is ensured as long as the proof of mixing employed also provides privacy.

Obviously, we are assuming that the elliptic ElGamal cryptosystem is semantically secure, so that no information is obtained from ballot ciphertexts.

Lemma 4.3.2. *If the proof of mixing provides privacy, our proposal also provides privacy.*

Proof. We are going to show that if an adversary $\mathcal{A}$ was able to obtain any information about the permutation π used by some voter v_i to generate her ballot, then $\mathcal{A}$ would be able to obtain the identity of a voter's vote in a mix-type voting system using that proof of mixing.

Let us assume that our proposal does not provide privacy. This means that an adversary $\mathcal{A}$ is able to obtain some information about some of the cleartexts in some components of vector C_{v_i}. Since the elliptic ElGamal cryptsystem is assumed to be semantically secure, this information can only consist of information on permutation π. Let $Alg(C, C_{v_i}, ZKP_{v_i})$ be the algorithm that generates such information.

Under that assumption, any observer of a mix-type voting system employing the above proof of mixing would obtain information about some of the mixed ballots as follows. Let C be the set of ballots cast by the voters, let C' be the set of mixed ballots and let ZKP be the corresponding proof of correct mixing. The observer would run $Alg(C, C', ZKP)$ and from the information obtained about the permutation π, it could link some ballots in C' with those in C, which are linked to the identity of the voters who cast them. After the decryption of the ballots in C', the observer could link the cleartext votes of some ballots in C' with the identity of some voters, and the privacy of the voting system would be compromised. Hence, the proof of mixing would not provide privacy and the claim follows.

□

Notice that, with our proposal, ballots whose array equals that of a previously cast ballot are not allowed. This fact, together with the *shuffling parameter awareness* of the proof of mixing, prevents message relation attacks [Pen11b, MS11] against the privacy of the system.

Chapter 5

Blind signature paradigm

This chapter begins with a brief summary of previous work on the blind signature paradigm for remote voting. Some of the open challenges to be addressed will be highlighted. These challenges have motivated our proposal detailed in Section 5.2.1, which was published in [MSV13].

In Section 5.3, the problem of anonymity revocation in case of double voting is presented. Several proposals are analyzed in depth in order to highlight their security flaws. A new proposal using e-coins to solve the double voting issues [MSV14] is detailed in Section 5.4.

5.1 Paradigm description

With this paradigm, the ballots are cast anonymously without an attached voter's signature. In this way, the need for a mechanism that breaks the link between a ballot and the voter casting it disappears. Instead, each ballot comes with a digital signature computed by a trusted party, the *Authentication Server*. Even though the ballot is not linked to the identity of its caster, that identity could be disclosed by tracing the source address of IP datagrams. To avoid that, the ballots are sent through an anonymous channel which keeps source addresses secret.

The first electronic voting scheme using an anonymous channel was proposed in [Cha88]. Later, [OMA+99] presented a proposal using blind signatures. The inclusion of an anonymous channel and the properties of blind signatures slightly changed the entities required to provide authentication and unicity.

Apart from the entities that were introduced in Section 1.1 (Voters, Electoral Roll Authority, Vote Tallying Authorities), this paradigm requires an additional entity that handles the authentication of the voters. We will refer to this entity as the *Authentication Server*. When contacted by a voter, it checks that the voter appears in the electoral roll and then computes a blind signature of her ballot. That blindly signed ballot will then be cast by

the voter through an anonymous channel. Hence, the Authentication Server guarantees that all the ballots received by the Ballot Collection Authority are cast by eligible voters. It also prevents double voting by checking that voters have not voted before.

The blind signature paradigm includes an additional phase in which ballots are blindly signed prior to being cast. A voting scheme implementing the blind signature paradigm consists of the following phases:

1. **Setup:** The vote tallying authorities create the election public key and publish it on a publicly accessible bulletin board. With this paradigm, the Authentication Server publishes its public key so that any entity can check its signatures. At the same time, the electoral roll authority publishes the electoral roll so that it can be modified in case of errors. The electoral roll contains the digital certificate of each eligible voter. The candidates' list as well as the instructions for the voters are also published.

   At the end of this phase, the bulletin board contains the election public key, the Authentication Server public key, the electoral roll and the candidates' list. From now on, these data can not be modified.

2. **Ballot authentication:** During this phase, each voter generates and encrypts her vote under the election public key. Next, the resulting ballot is blinded. The blinded ballot is signed by the voter and sent to the Authentication server.

   Upon receiving a signed blinded ballot, the Authentication Server authenticates the voter by verifying her digital signature and checks whether she is in the electoral roll. If this is the first time the voter contacts it, the Authentication Server proceeds by signing the blinded ciphertext and then sends the result back to the voter. The voter will unblind the data received, obtaining a valid digital signature on her ballot.

3. **Vote casting:** Once a voter has her ballot signed by the Authentication Server, she transmits it to the Ballot Collection Authority. This is done through an anonymous channel.

   Upon receiving a ballot, the Ballot Collecting Authority verifies that it has been properly signed by the Authentication Server. In such a case, the ballot and its signature are published on the bulletin board for its public verifiability.

   The vote casting phase ends when the voting period expires or when all the eligible voters have cast their ballots.

4. **Tallying:** When the vote casting phase has ended, the vote tallying authorities perform a verifiable decryption of each ballot in the bulletin

board. After that, the election result is published.

For usability purposes, the ballot authentication and the vote casting phases are performed in parallel. In this way, a voter can cast her ballot just after having had it signed.

An advantage of this approach is that it does not require the use of zero-knowledge proofs. If, after decrypting a ballot, the resulting vote turned out to have been incorrectly generated, it could simply be discarded without disrupting the process. Unfortunately, this paradigm provides only individual verifiability. A voter can verify that her vote has been counted but she has no evidence about the correctness of the rest of the ballots other than the signature from the authentication server. If the authentication server is completely trusted, public verifiability can be considered to be achieved to some extent.

The central component of this paradigm is the Authentication Server. This server must be a completely trusted party. If corrupted, it would be able to generate fake signed ballots and submit them anonymously. Since the only check performed by the polling station at vote reception is signature validation, all those ballots would be accepted as valid. Moreover, they would be indistinguishable from ballots cast by valid participants. This could be detected by forcing the Authentication Server to publish some proof of participant authentication (like a digitally signed request for the blind signature) and then checking that the amount of ballots collected does not exceed the number of blind signature requests. Unfortunately, after detecting this, the only solution would be to replace the corrupted server and repeat the election.

For this reason, the blind signature paradigm requires the Authentication Server to be trusted. This server also needs to be accessible throughout voting period.

5.1.1 Anonymous channel

Anonymity on the Internet is mostly achieved by using pseudonyms. Even though a pseudonym can not be directly related to the person behind it, the level of anonymity provided by this solution does not fulfill the anonymity requirements of an electronic election. This is because the identity of a vote caster could be obtained by tracing her IP address. In the blind signature-based paradigm, ballots are directly decrypted so that, after decryption, the source IP address could be linked to the cleartext of the vote. Taking that into account, the blind signature paradigm requires an anonymous channel [DDS09], preventing dishonest parties from tracing the source address of IP datagrams.

Anonymizing services can be implemented in two different ways:

- *Centralized:* The messages are sent to a server. This server (also known as the anonymizer) takes a set of received messages, adds some random messages to them, and then shuffles the resulting set. These systems usually require the sender to encrypt the message under an encryption algorithm that allows re-encryption. At the end, it will send the mixed and re-encrypted messages to the appropriate receivers.

- *Distributed:* A distributed anonymizer provides anonymity through a network of routers which randomize and send the messages to other peers of the network, following a protocol guaranteeing that each message will finally arrive at its intended receiver. It provides a higher degree of security than the centralized approach.

With security being one of the most important requirements for remote electronic voting, the use of distributed anonymous channels instead of centralized ones is highly recommended. These distributed anonymous channels can also be classified in two types:

- Low-latency [DMS04]: The main property of these anonymous channels is the fact that the messages are not delayed. The most well-known proposals are based on a technique called onion routing. With this technique, a circuit of routers is defined between the sender and the responder. The messages are encrypted in a way that every router can add or subtract a layer of encryption. Since the communication is not delayed, adversaries who can monitor the input and output of the anonymous channel are able to find traffic patterns which can be used to break the anonymity.

- High-latency [RMPAFD14]: These anonymous channels do not establish a bi-directional circuit between the communicating nodes. Instead, this solution is based on the idea of grouping the messages to mix and send them. Usually, a set of mixes delays and reorders messages to provide anonymity between the input and output messages. This solution provides more guarantees of anonymity than low-latency ones. However, it is not suitable for interactive services where delays affect the quality of service.

Given that electronic voting is not a fully interactive service and the importance of preserving voters' privacy, the most commonly used anonymous channels for the blind signature paradigm in remote electronic voting are the high-latency ones.

5.1.2 Authentication Server: trust and availability

Remember that the Authentication Server ensures that only ballots cast by eligible voters will be accepted by the Ballot Collection Authority and

prevent voters from casting more than one ballot. Both requirements must be fulfilled by a voting platform, hence the Authentication Server must be trusted. Voters require that server to be available so that it can blindly sign their ballots. Hence, it should be available throughout the election procedure. Trust in the authentication server can be increased, as proposed in [HMP95], by splitting it into a set of n entities which are unlikely to collude (for instance, each political party may be in charge of one). After that, by employing a blind multi-signature scheme, each entity is in possession of a secret key share so that computing a blind signature over an encrypted vote requires the participation of each of them. These entities are assumed to work properly at signing participants' valid votes but none of them would accept to participate in the generation of fake votes benefiting the opponent's choice. Having a distributed Authentication Server, whose components have to all be available throughout the voting period, increases the risk that some of them become out of order due to technical failures, denial of service attacks or intentional malfunctioning. Such a situation would play havoc with the election. Replacing the multi-signature scheme with a threshold signature one, like in [JLL02], provides some fault tolerance, but the trust in the server decreases, since it could act dishonestly without the need to corrupt all its components.

The fact that blind signatures are computed over already composed votes forces them to be generated during the voting period (participants cannot be required to make up their decision before), so this entity has to be available during this period. Any blind signature-based voting platform deployment needs to provide trust (requiring the Authentication Server to be split into a large amount of entities) and system availability (better achieved with a reduced server), so that a solution fulfilling both requirements may not be achievable.

5.2 Credential-based protocols

Instead of blindly signing the ballots, there is another approach consisting of the generation of a credential for each voter. This approach was first presented in [Rad95]. It proposed a variant of the blind signature paradigm in which the voters have to go to a voting authority office (acting as the Authentication Server) and ask for a pseudonym. Later, each voter will attach her blindly signed pseudonym (credential) to her vote and will send it to the polling station through an anonymous channel. If some voter tried to cheat by casting two or more votes, her identity could be disclosed. With this proposal, the vote is not blindly signed. Instead, a credential which has been blindly signed by the Authentication Server is needed. By means of this credential, the voter can demonstrate that she appears in the electoral roll. This approach increases the robustness of the protocol to attacks against the

Authentication Server because the blind signature can be performed before the election begins.

In [MV98], the need for the voters to obtain a pseudonym at the authority office is removed. With this proposal, a participant in an election authenticates to the Authentication Server, which provides her with a blindly signed voting ticket. After that, the voter will be able to decide and generate her vote, authenticate it employing the voting ticket and cast it at the polling station. In case some participant voted twice, the proposal provides a protocol for anonymity revocation. Unfortunately, [MV98] and some improvements on it [LHC03, YLY04, HWH05, RHOAGZ07, AJ09, BMA+11], have all turned out to be weak because they do not provide all the security properties claimed. It is worth noting that revoking double voters' anonymity may not be necessary, since a solution permitting such a situation to be detected and the votes involved eliminated would be enough in most cases.

A similar solution is presented in [AS08], in which the participants receive a blindly signed anonymous credential from the authentication server. This credential is attached to the vote cast, so that the polling station can check and store it in order to detect double voting. With this solution, a malicious polling station could take any vote-credential pair and replace its vote with a fake one. This fact could only be detected by the participant involved, who would have to complain. Consequently, this solution does not provide the universal verifiability property.

The proposal in [LLCC06] uses a distributed authentication server that issues voting tickets. Although the Authentication Server is distributed, the private keys are generated by a central dealer who, in case of corruption, could issue fraudulent voting tickets. Moreover, like in [AS08], the voting ticket is not verifiably linked to the vote cast, so that universal verifiability is not provided here either.

With these solutions, participants can contact the Authentication Server prior to deciding the content of their votes. Hence, voter authentication can be undertaken before the voting period begins, so that it is possible to have a longer authentication period that makes the system more reliable, since there is time to solve possible Authentication Server malfunctioning (especially when it has been distributed among a large set of entities).

With these proposals, it is possible that some participants that requested a voting credential in advance (maybe some days before the beginning of the voting period) decide not to cast a ballot. This would enable a corrupted Authentication Server to check the difference between credentials requested and votes cast, and, close to the end of the voting period, generate and send such an amount of fake votes. Since this malicious behaviour would remain undetected, any solution implementing anticipated blind signature computation strongly requires a distributed and completely trusted Authentication Server.

In [HS98], the use of a blindly certified public key as a voting credential is proposed, but the use of a non-distributed authentication server makes it vulnerable to the aforementioned attack.

5.2.1 New Scheme

A new blind signature-based e-voting system permitting anticipated interaction with the authentication server is presented below. With our proposal [MSV13], the participants obtain a voting credential by generating an anonymous private/public key pair that is blindly (multi)signed by the authentication server. During the voting period, each participant will cast her vote together with a digital signature computed under her anonymous certified key. The resulting system provides universal verifiability (not provided in [LLCC06, AS08]), double voting detection (so that double-cast ballots can be discarded) and the possibility of deploying a highly trusted distributed authentication server (not considered in [HS98]).

Entities

The entities involved in the system are the ones described in Section 5.1. Moreover, a new entity is required to issue certificates and some of the original ones have acquired new functionalities:

- *Certificate Authority (CA)*: Completely trusted external entity that issues public key certificates to citizens and institutions. It is not part of the voting platform but its certificates are needed for remote authentication.

- *Key Storage Trusted Party (KSTP)*: Like in [OMA+99], we assume the votes are cast encrypted under some public key cryptosystem whose private key is distributed employing some threshold scheme. This entity (or set of entities) is in charge of storing the secret key material required to decrypt votes. Our scheme could also accommodate the approach [FOO93], in which a participant first sends her encrypted vote and, once the voting period has concluded, she sends the decryption key. In this latter case, no trusted entity for storing decryption keys is needed.

- *Authentication Server (AS):* Entity that authenticates voters, checks the electoral roll and provides an anonymous voting credential (consisting of a blindly signed public key certificate) to each of them. With our proposal, this is a distributed entity composed of n entities $\{AS_1, \ldots, AS_n\}$. A voting credential can only be issued if every component of the Authentication Server participates.

- *Voters:* People listed in the electoral roll are those able to cast a vote. Each voter first authenticates and asks the Authentication Server for a voting credential. Afterwards, during the voting period, she will generate and cast a vote. We assume that each participant has a personal public key certified by the Certificate Authority.

- *Polling station (PS):* During the voting period, it receives encrypted ballots cast by voters. At the end of the voting period, it asks the KSTP to verifiably decrypt ballots and tally them.

Setup

This stage is devoted to the generation of public key material. New key material has to be generated for each election and should never be reused.

The entities composing the AS, $\{AS_1, \ldots, AS_n\}$, set up a multi-signature scheme. Each AS_i generates its private/public key pair (x_i, y_i). After that, the collective public key y is generated from $y_1, \ldots, y_n$. Finally, the set of keys $\{y_1, \ldots, y_n, y\}$ is certified by the CA.

In a similar manner, the KSTP generates a public key, Y, so that its secret key is distributed among the entities composing it. Public key Y, which will be employed for vote encryption, is also certified by the CA.

From now on, we will assume that all entities verify certificate validity prior to making use of these public keys.

Voting credential request

A voter $\mathcal{P}$, owning a private/public key pair $(x_\mathcal{P}, y_\mathcal{P})$ with its public key certificate $Cert_{CA}(y_\mathcal{P}, \mathcal{P})$, requests a voting credential as follows:

1. $\mathcal{P}$ generates a private/public key pair (x_t, y_t) at random.

2. $\mathcal{P}$ generates a public key certificate data structure, $M = CertData(y_t, \emptyset)$, for public key y_t with the information about its owner left blank.

3. $\mathcal{P}$ generates a random blinding factor r and computes $M' = Blind_r(M)$.

4. $\mathcal{P}$ sends M' its digital signature $Sign_{x_\mathcal{P}}(M')$ and $Cert_{CA}(y_\mathcal{P}, \mathcal{P})$ to each authentication server component AS_i.

5. Each AS_i verifies the validity of certificate $Cert_{CA}(y_\mathcal{P}, \mathcal{P})$ and the digital signature $\mathcal{P}$ on M' (using public key $y_\mathcal{P}$).

6. Each AS_i checks that $\mathcal{P}$ appears in the electoral roll and that she has not yet requested a voting credential.

7. Each AS_i computes the blind signature $\sigma'_i = Bsign_{x_i}(M')$ and publishes
$$(M', Sign_{x_\mathcal{P}}(M'), Cert_{CA}(y_\mathcal{P}, \mathcal{P}), \sigma'_i)$$
on its publicly accessible bulletin board.

8. $\mathcal{P}$ receives the signature σ'_i from each AS_i, unblinds it $\sigma_i = Ublind_r(\sigma'_i)$ and checks $Verif(M, \sigma_i, y_i)$.

9. $\mathcal{P}$ composes σ from $\{\sigma_1, \ldots, \sigma_n\}$, which is a signature on $CertData(y_t, \emptyset)$ verifiable under public key y.

10. $\mathcal{P}$ appends σ to $CertData(y_t, \emptyset)$, obtaining $Cert_{AS}(y_t, \emptyset)$. The resulting credential can be viewed as an anonymous public key certificate.

The resulting structure $Cert_{AS}(y_t, \emptyset)$ is a certificate on public key y_t blindly issued by the AS. With our proposal, this certificate (and knowledge of secret key x_t) is the *voting credential* that will permit $\mathcal{P}$ to cast her vote.

Any external verifier can check that the request comes from a citizen listed in the electoral roll by verifying the signed voting credential request. The response given by each AS_i can also be checked by verifying the signature σ'_i on M' under public key y_i. We conclude that the voting credential request procedure is universally verifiable.

Since all partial signatures σ'_i issued by AS_i are verifiable under the same public key y_i, they can be verified in batch.

Voting

When the voting period begins, each participant $\mathcal{P}$, who previously requested a voting credential, decides her vote and casts it. The following procedure is performed:

1. $\mathcal{P}$ composes her vote m.

2. $\mathcal{P}$ encrypts m under public key Y, obtaining $C = E_Y(m)$.

3. $\mathcal{P}$ computes the signature $Sign_{x_t}(C)$.

4. $\mathcal{P}$ casts her vote by sending the tuple
$$(C, Sign_{x_t}(C), Cert_{AS}(y_t, \emptyset))$$
to the polling station through an anonymous channel.

5. Upon reception, the polling station (PS) verifies that $Cert_{AS}(y_t, \emptyset)$ is a valid certificate (by verifying its signature by the AS).

6. PS verifies the signature $Sign_{x_t}(C)$ on C using public key y_t.

7. PS also checks that the certificate $Cert_{AS}(y_t, \emptyset)$ has not been used together with any previous vote.

8. PS publishes the tuple received on the public list of received votes.

Any external entity wishing to verify the process simply has to carry out the same checks performed by the polling station, so that vote casting is universally verifiable.

The computational cost of this phase can be reduced by verifying a set of the voting credentials from different participants (step 5) in batch, since all verifications are to be performed using the public key of the AS.

Tallying

At the end of the voting period, the components of the KSTP make their secret key material public so that any entity is able to decrypt all the ballots received by herself (making the process verifiable). The decrypted votes are tallied and the results are finally published.

5.2.2 Security

The security analysis holds on the assumption that the Authentication Server is distributed in such a way that its components will not collaborate to issue fraudulent credentials.

Integrity

Integrity is the hardest security property to fulfill when dealing with a voting scheme using the blind signature paradigm. There are three main threats to address:

- *Authentication*: Every ballot received has to be cast by a valid voter. With this scheme, the Authentication Server asks the participants to authenticate themselves prior to issuing a blind signature on their anonymous public key certificate. By authenticating them, it can check whether they appear in the electoral roll and that they have not requested a voting credential before.

  By making the signed requests publicly available, any entity is able to verify that each request comes from a valid participant. Also, by publishing the blind signature σ'_i computed over M', each AS_i can defend against participants claiming that their request was not served.

- *Vote modification*: No vote can be modified during the voting process. In this case, the protocol ensures that the vote cast by a voter carries a digital signature verifiable under a public key that is certified by the

AS. Any manipulation of the vote would cause a signature validation failure.

- *Unicity*: A voter cannot cast more than one ballot. Under the assumption that the AS cannot misbehave, a participant in the electoral roll cannot obtain more than one voting credential. Hence, if a voter casts more than one vote, she would be using the same credential. This fact is easy to detect, since both voting tuples will share the same certificate issued by the AS.

  An appropriate policy should be implemented to deal with these cases:

  - Invalidate multiple ballots by the same (anonymous) voter
  - Consider only the last one cast (this policy allows vote changing).

Privacy

The use of a blind signature scheme ensures that the voting credential obtained by a participant cannot be related with the data that she sends to request it. In this way, the voting credential is anonymous. Sending the ballot through an anonymous channel ensures that the source of votes received by the polling station cannot be traced.

Verifiability

Given the bulletin board and the electoral roll, which are public, anyone can check that all the ballots have been cast together with a valid credential signed by the Authentication Server. This ensures the authentication property as well as the unicity property, as explained before.

Apart from that, all the ballots on the bulletin board are verifiably decrypted in order to obtain the results of the election. In this way, any voter can check that her ballot has been counted in the final tally and she can even check that the content of her ballot was correctly decrypted. These two validations ensure for any voter that the ballot has been correctly cast (cast-as-intended). Any auditor can check the correctness of the decryptions and the validity of the certificates. Hence, our solution provides universal verifiability.

5.3 Double voting perception

Designing a secure electronic voting scheme which uses an anonymous channel is challenging work. The solution proposed in Section 5.2.1 manages to fulfill the security requirements, also offering good performance. However, in some scenarios it could be necessary to identify a misbehaving voter. Given that privacy is mandatory, the opportunity to provide privacy up to

a certain point in which the identity of a voter can be disclosed opens a window of possibilities. Taking that into account, several proposals have been presented in recent years.

5.3.1 Mu and Varadharadjan scheme

The first approach to handle double voting with anonymity revocation was presented in [MV98]. It proposed an improvement over the variant of [Rad95], in which the interaction between the Authentication Server and the voters can be carried out remotely. The proposal of [MV98] involves five parties: the voters, the Authentication Server (AS), voting servers (VS) that collect voting tickets from the voters, a ticket counting server (TCS), and trusted certificate authorities. The protocol is composed of four steps:

Setup

During this phase, the AS generates an RSA modulus n_{AS} (the product of two large primes) and a public/private key pair e_{AS}, d_{AS}. Each voter V_i also generates an RSA modulus n_{V_i} and her own pair of RSA keys e_{V_i} and d_{V_i}. As the system also uses a variant of ElGamal signature scheme, a finite field $\mathbb{F}_p$, with p being a large prime, is also generated. After this information has been generated, a voter can cast a ballot as described below.

The anonymous ticket acquiring phase

A voter V_i chooses a blind factor $b \in \mathbb{Z}_{n_{AS}}$, and picks up three random numbers $g \in \mathbb{F}_p$ and $r, k_1 \in [1, \ldots, p-1]$. After that, V_i prepares the following parameters to request an anonymous ticket:

$$a = g^r \mod p, \quad w_1 = gb^{e_{AS}} \mod n_{AS},$$

$$w_2 = g^{k_1} b^{e_{AS}} \mod n_{AS}, \text{ and } w_3 = ab^{e_{AS}} \mod n_{AS}.$$

The voter V_i sends $\{Cert_{V_i}, (w_1||w_2||w_3||t)^{d_{V_i}} \mod n_{V_i}\}$ to the AS. Value t denotes a timestamp.

Upon receiving the request, the AS first verifies the voter's certificate and validates the voter's signature $(w_1||w_2||w_3||)t^{d_{V_i}}$. If the verification succeeds, the AS chooses a random number $k_2 \in [1, \ldots, p-1]$, and computes:

$$w_4 = (k_2||t)^{e_{V_i}} \mod n_{V_i},$$

$$w_5 = (w_1^{3k_2} w_2^2 w_3)^{d_{AS}} = (y_1 y_2 a)^{d_{AS}} b^{3(k_2+1)} \mod n_{AS},$$

with $y_1 = g^{k_1+k_2}$ and $y_2 = g^{k_1+2k_2}$. Then, the AS stores k_2 and V_i's identity in its database. The parameter k_2 is different for each voter, so that it can be used to trace the owner of a ballot if the voter casting it voted more than once. Finally, the AS sends message $\{w_4, (w_5||t)^{e_{V_i}} \mod n_{V_i}\}$ to V_i.

After that, V_i decrypts w_4 to obtain k_2 , and computes $y_1 = g^{k_1+k_2}$ and $y_2 = g^{k_1+2k_2}$. Next, V_i removes $b^{3(k_2+1)}$ from w_5 to obtain the signature $s = (y_1 y_2 a)^{d_{AS}} \mod n_{AS}$. Then V_i computes

$$s_1 = (k_1 + k_2)^{-1}(ma - r) \mod p - 1,$$

$$s_2 = (k_1 + 2k_2)^{-1}(ma - r) \mod p - 1,$$

with m being V_i's voting intention in clear form. The voting ticket T is constructed as $T = \{a||g||y_1||y_2||s||s_1||s_2||m\}$.

The voting and ticket collecting phase

After generating the anonymous ticket in the previous phase, V_i can cast the voting ticket T with the Voting Server as follows:

1. V_i sends $(T||t)^{e_{VS}} \mod n_{VS}$ to the VS.

2. Upon receiving the message, the VS decrypts it to obtain the voting ticket T. Then, the VS verifies the AS's signature by checking whether

   $$s^{e_{AS}} = (y_1 y_2 a) \mod n_{AS}.$$

   In that case, the VS further validates the correctness of the signatures s_1 and s_2 by checking whether $y_1^{s_1} a = g^{ma} \mod p$ and $y_2^{s_2} a = g^{ma} \mod p$.

If all the verifications succeed, the VS accesses its database to check whether the entry $\{a, g, y_1, y_2\}$ already exists. If it does not, it accepts the voting ticket. Finally, the VS sends all the voting tickets in a batch to the TCS.

The ticket counting phase

When the voting period has ended and all VS's have sent their voting tickets, the TCS checks whether there are two voting tickets

$$T = \{a||g||y_1||y_2||s||s_1||s_2||m\} \text{ and } T' = \{a'||g'||y_1'||y_2'||s'||s_1'||s_2'||m'\}$$

such that $(a, g, y_1, y_2) = (a', g', y_1', y_2')$. If some duplicate entry is found, the TCS can find the identity of the double-voting voter by computing

$$k_1 + k_2 = (m'a - ma)/(s_1' - s_1) \mod p - 1,$$

and

$$k_1 + 2k_2 = (m'a - ma)/(s_2' - s_2) \mod p - 1.$$

From the above two equations, TCS can obtain k_2 and identify the voter. After validating all the tickets, the TCS tallies the tickets and announces the result.

Security issues

The proposal of Mu and Varadharadjan [MV98] was proven to be insecure when several security flaws were shown [CJT03, LHC03, YLY04]. These flaws allowed attackers to forge valid anonymous tickets without being authenticated. The attack is as follows.

The attacker chooses three random numbers k_1, k_2 and $r \in [1, \ldots, p-1]$ and computes:

$$a = g^{re_{AS}}, \;\; y_1 = g^{k_1 e_{AS}}, \;\; y_2 = g^{k_2 e_{AS}},$$

$$s = g^{k_1+k_2+r} \mod n_{AS},$$

$$s_1 = (k_1 + k_2)^{-1}(ma - r) \mod p - 1,$$

$$s_2 = (k_1 + 2k_2)^{-1}(ma - r) \mod p - 1.$$

Next, the attacker can construct the ticket $T = \{a||g||y_1||y_2||s||s_1||s_2||m\}$. This ticket will pass the verifications of VS and TCS with very high probability, since it passes the signature verification and the probability of ticket collision is very low. Moreover, since the attacker can arbitrarily choose the parameters, it can forge many distinct valid tickets without being detected.

5.3.2 Lin et al. scheme

In [LHC03], an improvement of [MV98] is proposed. The phases remain the same but the system is slightly changed in order to be secure against ticket forgery:

The anonymous ticket acquiring phase

A voter V_i computes w_1 and w_2:

$$w_1 = g^r b_1^{e_{AS}} \mod n_{AS} \text{ and } w_2 = g^k b_2^{e_{AS}} \mod n_{AS}$$

with b_1, b_2 being two blind factors, $r, k_1 \in [1, \ldots, p-1]$ being randomly chosen by the voter and $g \in \mathbb{F}_p$ being the system's public parameter. Then, the voter sends $\{Cert_{V_i}, t, w_1, w_2, ((w_1||w_2||t)^{d_{V_i}} \mod n_{V_i})\}$ to the AS.

The AS verifies the certificate and checks the correctness of the voter's signature $((w_1||w_2||t)^{d_{V_i}} \mod n_{V_i})$. If the verification succeeds, the AS chooses a random number $k_2 \in [1, \ldots, p-1]$ and computes:

$$w_3 = (k_2||t)^{e_{V_i}} \mod n_{V_i}$$

$$w_4 = w_1^{d_{AS}} = a^{d_{AS}} b_1 \mod n_{AS},$$

$$w_5 = (w_2 \cdot g^{k_2})^{d_{AS}} = y_1^{d_{AS}} b_2 \mod n_{AS},$$

$$w_6 = (w_2^2 \cdot g^{k_2})^{d_{AS}} = y_2^{d_{AS}} b_2^2 \mod n_{AS},$$

with $a = g^r$, $y_1 = g^{k_1+k_2}$, and $y_2 = g^{2k_1+k_2}$. The AS stores k_2 together with the identity of V_i in its database and sends $\{w_3, (w_4||w_5||w_6||t)^{e_{V_i}} \mod n_{V_i}\}$ to V_i.

V_i obtains k_2 from w_3, and calculates y_1 and y_2. Next, V_i generates s_1, s_2 and s_3, removing the blind factors as follows:

$$s_1 = w_4 \cdot b_1^{-1} = a^{d_{AS}} \mod n_{AS},$$

$$s_2 = w_5 \cdot b_2^{-1} = y_1^{d_{AS}} \mod n_{AS},$$

$$s_3 = w_6 \cdot b_2^{-2} = y_2^{d_{AS}} \mod n_{AS}.$$

After that, the voter V_i signs the vote content m using ElGamal digital signature scheme. Let $x_1 = k_1 + k_2$ and $x_2 = 2k_1 + k_2$ be the secret keys of the ElGamal cryptosystem, with y_1 and y_2 being the corresponding public keys. The voter generates the signatures of message m: (a, s_4) and (a, s_5) as follows:

$$s_4 = x_1^{-1}(ma - r) \mod p - 1,$$

$$s_5 = x_2^{-1}(ma - r) \mod p - 1.$$

Finally, the voter V_i generates the ticket $T = \{s_1||s_2||s_3||s_4||s_5||a||y_1||y_2||m\}$.

The voting and ticket collecting phase

After generating the anonymous ticket in the previous phase, V_i casts the voting ticket T with the Voting Server as follows:

1. V_i sends T to the VS.
2. The VS verifies whether a, y_1 and y_2 are valid by checking the following equalities:
$$a = s_1^{e_{AS}} \mod n_{AS},$$
$$y_1 = s_2^{e_{AS}} \mod n_{AS},$$
$$y_2 = s_3^{e_{AS}} \mod n_{AS}.$$
3. If everything is correct, the VS verifies the correctness of signatures (a, s_4) and (a, s_5) on m by checking:
$$g^{ma} = y_1^{s_4} a \mod p,$$
$$g^{ma} = y_2^{s_5} a \mod p.$$
4. The VS checks that it has not stored a previous entry $\{a, y_1, y_2\}$.

If the verifications succeed, the VS sends all the voting tickets in a batch to the TCS.

The ticket counting phase

After receiving all the tickets from the VS, the TCS runs a double voting detection process:

1. The TCS checks the existence of a double voting by locating a duplicate entry (a, y_1, y_2) in T.
2. If a duplicate entry exists with a different vote, the TCS can find the identity of the double-voting voter by computing
$$x_1 = (m'a - ma)/(s'_4 - s_4) \mod p - 1,$$
and
$$x_2 = (m'a - ma)/(s'_5 - s_5) \mod p - 1.$$
3. The TCS can compute $k_1 = x_2 - x_1 = (2k_1 + k_2) - (k_1 + k_2)$. Now, it is easy to obtain $k_2 = x_1 - k_1$; hence, the TCS can find the identity of the illegal voter.

After validating all the tickets, the TCS tallies the tickets and announces the result.

Security issues

An attack on [LHC03] was presented in [HWH05]. The attack allows the AS to identify the owner of any ticket cast. Assuming that the TCS publishes all the tickets cast, the AS could trace the identity of the sender of a ticket $T_i = \{s_1||s_2||s_3||s_4||s_5||a||y_1||y_2||m\}$ belonging to voter V_i. The AS would proceed as follows:

1. Compute $x = (y_2/y_1) = g^{k_1} \mod p$.
2. Select a record (V', k'_2) from the database and check
$$x \cdot g^{k'_2} = y_1,$$
$$x^2 \cdot g^{k'_2} = y_2.$$
if the previous equalities hold, the identity of V_i is V'.
3. Repeat step 2 until a database record satisfies these equalities.

Following this easy procedure, the AS can identify all the voters and their votes. Hence, the proposal fails on providing anonymity.

5.3.3 Hwang et al. scheme

The scheme proposed in [HWH05] is an improvement on the broken scheme [LHC03]. It simply adds a few more computations. For simplicity, we just detail the additional or modified computations.

Anonymous ticket acquiring phase

A voter V_i computes w_1, w_2, w_1', w_2' :

$$w_1' = h^r b_1^{e_{AS}} \mod n_{AS} \text{ and } w_2' = h^k b_2^{e_{AS}} \mod n_{AS}$$

with $h \in \mathbb{F}_p$ being another public parameter of the system so that $h \neq g$. The voter sends $\{Cert_{V_i}, t, w_1, w_1', w_2, w_2', ((w_1||w_1'||w_2||w_2'||t)^{d_{V_i}} \mod n_{V_i})\}$ to the AS.

The AS verifies the certificate and voter's signature $((w_1||w_1'||w_2||w_2'||t)^{d_{V_i}} \mod n_{V_i})$. If the verification succeeds, the AS chooses a random number $k_2 \in \mathbb{F}_p$ and computes w_3, w_4, w_5, w_6, w_7:

$$w_5 = (w_1')^{d_{AS}} = (a')^{d_{AS}} b_1 \mod n_{AS},$$

$$w_6 = (w_2 \cdot g^{k_2})^{d_{AS}} = y_1^{d_{AS}} b_2 \mod n_{AS},$$

$$w_7 = ((w_2')^2 \cdot h^{k_2})^{d_{AS}} = y_2^{d_{AS}} b_2^2 \mod n_{AS},$$

with $a' = h^r$, and $y_2 = h^{2k_1+k_2}$. The AS stores k_2 together with the identity of V_i in its database and sends $\{w_3, (w_4||w_5||w_6||w_7||t)^{e_{V_i}} \mod n_{V_i}\}$ to V_i.

V_i obtains k_2 from w_3, and calculates y_1 and y_2. Next, V_i generates s_1, s_2, s_3 and s_4 by removing the blind factors:

$$s_2 = w_5 \cdot b_1^{-1} = (a')^{d_{AS}} \mod n_{AS},$$

$$s_3 = w_6 \cdot b_2^{-1} = y_1^{d_{AS}} \mod n_{AS},$$

$$s_4 = w_7 \cdot b_2^{-2} = y_2^{d_{AS}} \mod n_{AS}.$$

After that, the voter V_i signs the vote m using the ElGamal digital signature scheme. Let $x_1 = k_1 + k_2$ and $x_2 = 2k_1 + k_2$ be the secret keys of the ElGamal cryptosystem, with $y_1 = g^{x_1}$ and $y_2 = h^{x_2}$. The voter generates the signatures of the message m: (a, s_5) and (a', s_6) as follows:

$$s_5 = x_1^{-1}(ma - r) \mod p - 1,$$

$$s_6 = x_2^{-1}(ma' - r) \mod p - 1.$$

The voter V_i generates the ticket $T = \{s_1||s_2||s_3||s_4||s_5||s_6||a||a'||y_1||y_2||m\}$.

The voting and ticket collecting phase

After generating the anonymous ticket in the previous phase, V_i can cast the voting ticket T with the Voting Server as follows:

1. V_i sends T to the VS.

2. The VS verifies whether a, a', y_1 and y_2 are valid by checking the following equalities:

$$a = s_1^{e_{AS}} \mod n_{AS},$$

$$a' = s_2^{e_{AS}} \mod n_{AS},$$

$$y_1 = s_3^{e_{AS}} \mod n_{AS},$$

$$y_2 = s_4^{e_{AS}} \mod n_{AS}.$$

3. If the previous equalities hold, the VS verifies the correctness of the signatures (a, s_5) and (a', s_6) on m by checking:

$$g^{ma} = y_1^{s_5} a \mod p,$$

$$g^{ma'} = y_2^{s_6} a \mod p.$$

4. The VS checks that it has not previously stored an $\{a, a', y_1, y_2\}$ entry.

After the voting time expires, the VS sends all the voting tickets in a batch to the TCS.

The ticket counting phase

After receiving all the tickets from the VS, the TCS runs a double-voting detection process:

1. The TCS checks whether double voting exists by locating a duplicate entry (a, a', y_1, y_2) in T.

2. If a duplicate entry with a different vote exists, the TCP can compute

$$x_1 = (m'a - ma)/(s_5' - s_5) \mod p - 1,$$

$$x_2 = (m'a' - ma')/(s_6' - s_6) \mod p - 1,$$

$$k_1 = x_2 - x_1,$$

$$k_2 = x_1 - k_1.$$

3. The TCS can find the identity of the illegal voter by asking the AS about k_2.

After validating all the tickets, the TCS tallies the tickets and announces the result.

Security issues

An attack on the anonymity of this scheme can be performed by the AS. Taking into account that the parameters w_1 and w_1' are blinded with the same blinding factor, when a voter sends

$$\{w_1, w_1', w_2, w_2', ((w_1||w_1'||w_2||w_2'||t)^{d_V}) \mod n_V\}$$

to the AS, it can compute the value $z_1 = w_1/w_1' = g^r/h^r$ and store it together with the voter's identity. In this way, when the TCS publishes all the tickets, the AS can compute the value $z_2 = a/a' = g^r/h^r$ for each ticket published and locate a z_1 with the same value in its database.

5.3.4 Rodríguez-Henríquez et al. scheme

Rodríguez-Henríquez et al. presented a new proposal [RHOAGZ07] based on [LHC03]. It offered better performance and security against the successful anonymity attacks for [HWH05]. These improvements are obtained by replacing ElGamal signature scheme with DSA. The system is modified to accommodate this replacement.

Setup

The setup remains the same, but some new parameters are added. Apart from the RSA key generation, it also requires the generation of DSA parameters: two large public primes p and q so that $q|(p-1)$, a generator $g \in \mathbb{Z}_{p-1}^*$, together with a value $\alpha = g^{(p-1)/q} \mod p$. A DSA private key x for voter V_i is also generated.

The anonymous ticket acquiring phase

A voter V_i generates two blind factors b_1, b_2, a random value $k_1 < q/3$ and her DSA private key $x \in_R \mathbb{Z}_{q-1}^*$. After that, she computes:

$$y = \alpha^x \mod p,$$

$$w_1 = y b_1^{e_{AS}} \mod n_{AS},$$

$$w_2 = \alpha^{k_1} b_2^{e_{AS}} \mod n_{AS},$$

and sends $\{Cert_{V_i}, t, w_1, w_2, ((w_1||w_2||t)^{d_{V_i}} \mod n_{V_i})\}$ to the AS.

The AS validates the voter's signature $((w_1||w_2||t)^{d_{V_i}} \mod n_{V_i})$ and verifies the certificate. If the verification succeeds, the AS chooses a random number $k_2 < q/3$ and computes w_3, w_4, w_5, w_6:

$$w_3 = (k_2||t)^{e_{V_i}} \mod n_{V_i},$$

$$w_4 = w_1^{d_{AS}} = y^{d_{AS}} b_1 \mod n_{AS},$$

$$w_5 = (w_2 \cdot \alpha^{k_2})^{d_{AS}} = (\alpha^{k_1+k_2})^{d_{AS}} b_2 \mod n_{AS},$$
$$w_6 = (w_2^2 \cdot \alpha^{k_2})^{d_{AS}} = (\alpha^{2k_1+k_2})^{d_{AS}} b_2^2 \mod n_{AS}.$$

The AS stores k_2 together with the identity of V_i in its database and sends $\{w_3, (w_4||w_5||w_6||t)^{e_{V_i}} \mod n_{V_i}\}$ back to V_i.

V_i obtains k_2 by decrypting w_3 and computes s_1, s_2, s_3 by removing the blind factors:

$$s_1 = w_4 \cdot b_1^{-1} = y^{d_{AS}} \mod n_{AS},$$
$$s_2 = w_5 \cdot b_2^{-1} = (\alpha^{k_1+k_2} \mod p)^{d_{AS}} \mod n_{AS},$$
$$s_3 = w_6 \cdot b_2^{-2} = (\alpha^{2k_1+k_2} \mod p)^{d_{AS}} \mod n_{AS}.$$

The voting and ticket collecting phase

Voter V_i casts her ballot consisting of message m signed under DSA. In order to do that, she computes:

$$x_1 = k_1 + k_2,$$
$$x_2 = 2k_1 + k_2,$$
$$r_1 = (\alpha^{x_1} \mod p) \mod q,$$
$$r_2 = (\alpha^{x_2} \mod p) \mod q,$$

and generates the signatures (r_1, s_4) and (r_2, s_5) with:

$$s_4 = x_1^{-1}(m + xr_1),$$
$$s_5 = x_2^{-1}(m + xr_2).$$

Additionally, voter V_i must compute:

$$l_1 = (\alpha^{k_1} \mod p) \cdot (\alpha^{k_2} \mod p) \mod n_{AS},$$
$$l_2 = ((\alpha^{k_1} \mod p)^2 \mod n_{AS}) \cdot ((\alpha^{k_2} \mod p) \mod n_{AS}).$$

The voting ticket T is generated as:

$$T = \{s_1, s_2, s_3, s_4, s_5, r_1, r_2, l_1, l_2, m\}$$

When the VS receives a ballot, it checks three digital signatures:

$$y \mod n_{AS} = s_1^{e_{AS}} \mod n_{AS},$$
$$l_1 = s_2^{e_{AS}} \mod n_{AS},$$
$$l_2 = s_3^{e_{AS}} \mod n_{AS}.$$

After that, the VS runs the DSA verification algorithm for pairs (r_1, s_4) and (r_2, s_5). If all the signatures are valid, the VS accepts and stores the ticket.

The ticket counting phase

When the voting phase ends, the VS sends all the tickets to the TCS. After receiving all the tickets, the TCS runs a double voting detection process:

1. The TCS checks whether double voting exists by locating a duplicate entry (y, r_1, r_2, l_1, l_2) in T.

2. If duplicate entries with different voting content exist, the TCS can find the double-voting voter's identity by computing

$$x_1 = (m' - m)/(s'_4 - s_4) \mod q,$$

$$x_2 = (m' - m)/(s'_5 - s_5) \mod q,$$

$$k_1 = x_2 - x_1,$$

$$k_2 = x_1 - k_1.$$

3. The TCS can find the identity of the illegal voter by cooperating with the AS, as it knows k_2.

After validating all the tickets, the TCS tallies the tickets and announces the result.

Security issues

A malicious voter can replace the original values of a_1, s_1 with y_1, s_3, respectively, and compute a new value for s_5 as $s_5 = r^{-1}(m \cdot y_1 - (k_1 + k_2)) \mod (p-1)$. In this way, she is able to create a forged ticket which passes all the validations. In a similar way, by replacing the values of a_2, s_2 with y_2, s_4, respectively, and computing $s_6 = r^{-1}(m \cdot y_2 - (2k_1 + k_2)) \mod (p-1)$, she can create another forged ticket.

5.3.5 Baseri et al. scheme

This scheme [BMA+11] is the last previous contribution to the paradigm. In this case, the amount of computations is reduced and the resulting system is resistant to all the previous attacks found against the paradigm.

Setup

The AS generates:

- $(e_{AS}, n_{AS}), d_{AS}$ and $(e'_{AS}, n'_{AS}), d'_{AS}$: Two RSA public/private key pairs, with $e_{AS} > e'_{AS}$.

- g, h public elements of prime order l in $\mathbb{Z}^*_{n_{AS}}$.

Each voter V_i has:

- $(e_{V_i}, n_{V_i}), d_{V_i}$: Her RSA public/private key pair.
- $Cert_{V_i}$: Certificate of V_i's public key.
- u_{V_i}: A unique value, only known by V_i.
- ID_{V_i}: The public identity of V_i, which corresponds to $g^{u_{V_i}}$ in $\mathbb{Z}^*_{n_{AS}}$.

The anonymous ticket acquiring phase

The voter V_i selects two blind factors b_1 and b_2 and computes:

$$A = g^{u_{V_i}} h \mod n_{AS},$$
$$A' = A^s \mod n_{AS},$$
$$B = g^{x_1} h^{x_2} \mod n_{AS},$$
$$w_1 = B b_1^{e'_{AS}} \mod n_{AS},$$
$$w_2 = (A' + B) b_2^{e_{AS}} \mod n_{AS},$$

with $x_1, x_2 \in_R \mathbb{Z}^*_{e'_{AS}}$ and $s \in_R \mathbb{Z}^*_{e_{AS}}$.

The voter sends $\{Cert_{V_i}, A, w_1, w_2, t, ((A||w_1||w_2||t)^{d_{V_i}}) \mod n_{V_i}\}$ to the AS.

The AS verifies the certificate and the correctness of the value A and the signature. After passing all the verifications, the AS computes:

$$w_3 = A^{d_{AS}} \mod n_{AS},$$
$$w_4 = w_1^{d'_{AS}} \mod n_{AS},$$
$$w_5 = w_2^{d_{AS}} \mod n_{AS},$$

and sends $\{((w_3||w_4||w_5||t)^{e_{V_i}}) \mod n_{V_i}\}$ to V_i.

Upon receiving it, V_i decrypts it and obtains w_3, w_4 and w_5, which is unblinded as follows:

$$s_1 = w_3^s \mod n_{AS},$$
$$s_2 = w_4 / b_1 \mod n_{AS},$$
$$s_3 = w_5 / b_2 \mod n_{AS}.$$

With all these values, V_i only needs to choose her vote and generate:

$$d = H(A', B, s_1, s_2, s_3, vote, nonce) \mod e_{AS},$$
$$r_1 = d u_{V_i} s + x_1 \mod e_{AS},$$
$$r_2 = ds + x_2 \mod e_{AS}.$$

Finally, the voting ticket

$$T = \{A', B, vote, s_1, s_2, s_3, d, r_1, r_2, nonce\}$$

can be sent to the VS.

The voting and ticket collecting phase

When the VS receives a ticket, it proceeds as follows:

- Verifies the signatures s_1, s_2 and s_3.
- Checks the equation: $g^{r_1}h^{r_2} = (A')^d B \mod n_{AS}$.

If the ticket passes these validations, the VS stores it in its database.

The ticket counting phase

Once the voting phase has ended, the VS sends its tickets to the TCS. The TCS verifies whether or not double voting has occurred. This is done by checking parameters A' and B of the tickets. If the TCS finds the same values A' and B on two or more tickets it will obtain the identity of the double voter by computing:

$$u_{V_i} = \frac{r_1 - r_1'}{r_2 - r_2'} \mod e_{AS},$$

$$ID_{V_i} = g^{u_{V_i}} \mod n_{AS}.$$

Finally, the TCS counts the valid tickets and publishes them in the bulletin board.

Security issues

The scheme of [BMA+11] consists of an RSA-type cryptosystem whose security holds on the assumed intractability of the *integer factorization* and *discrete logarithm* problems (see Section 2.1).

As pointed out in [MSV13], during the setup phase, the authentication server publishes two RSA keys sharing the same modulus $n = pq$, being p and q two large primes, and two elements g, h having the same large prime order l in $\mathbb{Z}_n^*$. Factoring n is assumed to be hard.

The failure is due to the publication of the large prime order elements g and h. Being g an order l element in $\mathbb{Z}_n^*$ means that $g^l \equiv 1 \pmod{n}$, which is only possible if l divides $\varphi(n) = (p-1)(q-1)$. Since l is a large prime, it will probably divide $(p-1)$ or $(q-1)$, but not both of them. In the former case, $g^l \equiv 1 \pmod{n}$ implies that $g^l \equiv 1 \pmod{p}$ and $g^l \equiv 1 \pmod{q}$. Since l probably does not divide $(q-1)$, $g^l \equiv 1 \pmod{q}$ is only possible if $g \equiv 1 \pmod{q}$, so that $q = \gcd(g-1, n)$. This proves that n is easily factored from public parameters by simply performing a *greatest common divisor* (gcd) computation. A description of similar failures on RSA-type cryptosystems is presented in [ML98].

5.4 E-coins in e-voting

As mentioned in the previous section, the solutions proposed to achieve double voting anonymity revocation in the blind signature paradigm do not fulfill the security requirements of electronic voting.

The anonymity environment provided by the anonymous channel in the blind signature paradigm is somehow similar to the scenarios in which e-coins are used. Moreover, the security properties and the protocols used by e-coin systems make them suitable for use in electronic voting in case of double voting.

In this section, a new construction presented in [MSV14] is going to be detailed. The construction uses e-coins to validate the voters and is able to provide anonymity disclosure in case of double spending.

5.4.1 E-coins

E-coin systems provide an anonymous way to implement payments on the Internet. The participants involved in such a system are:

- *Bank*: Entity that provides e-coins to payers.
- *Payer*: Entity that asks the bank for e-coins that will later be spent with some merchant.
- *Merchant*: Entity receiving payments that will then deposit the e-coins received to the bank.

In such systems, the payer authenticates with the bank and obtains an e-coin through the *withdrawal* protocol. This protocol ensures that the bank obtains no information about the e-coin generated so that it can later be spent anonymously. E-coins are spent with the merchant through the *payment* protocol. Since e-coin systems are anonymous, it is mandatory that a dishonest payer cannot forge new valid coins or spend the same e-coin more than once.

An e-coin system must provide the following security properties:

- *Independence*: The security of the system does not depend on users, time, medium or location.
- *Unforgery*: Coins cannot be created without the bank's collaboration.
- *Untraceability*: It is not possible to identify the honest spender of an e-coin.

E-coin systems prevent double-spending in different ways. With *online* systems, prior to accepting a payment, the merchant contacts the bank and checks that the e-coin received has not yet been spent. On the other hand,

with *offline* systems the payment protocol does not involve the bank. Such systems require anonymity revocation in case of double spending so that, if some e-coin is spent twice, the identity of the dishonest payer becomes disclosed.

In the literature, there are several offline e-coin systems [Bra94, LTW05] in which the payment protocol follows a three-move format. Given an e-coin, the payer (whose identity is kept secret) first sends a commitment message CMT to the merchant, containing some data that was blindly signed by the bank during the withdrawal protocol. By verifying this signature, the merchant is sure the e-coin received was issued by the bank, then the merchant returns a random challenge CH and the payer provides a proper response RSP. Computing this response requires knowledge of some private key material only known by the person who withdrew that coin. A dishonest payer spending the same e-coin twice will have to provide two valid responses to two different challenges. Some offline e-coin protocols are designed so that the identity of the double-spender can be obtained from a pair of responses to two different challenges related to the same e-coin. In other proposals, anonymity revocation is carried out by a trusted authority.

There are a strong similarity between the security requirements of e-voting and e-coin systems. In an election, a voter can vote only once while an e-coin cannot be spent more than one time. With e-voting, the identity of a voter cannot be linked to the vote she cast, while e-coin systems do not make it possible to know the identity of the person who spent a given e-coin. Moreover, in the remote voting paradigm based on blindly signed credentials, if a voter voted several times, her anonymity would be lifted, as occurs in offline e-coin systems in case of double-spending. The unforgery property of e-coins is also required for voting credentials, otherwise dishonest entities could forge them and cast invalid votes that would be considered valid.

5.4.2 E-voting scheme using e-coins

In this section, our proposal published in [MSV14] is detailed. More precisely, it is shown how a (three-move format) offline e-coin protocol can be employed to construct a blindly signed credential-based e-voting system.

Entities

The following entities are involved in the protocol:

- *Voter*: Person listed in the electoral roll. In our construction, she plays a role similar to that of a payer in an e-coin system.
- *Authentication server*: Authenticates participants by checking whether they appear in the electoral roll and blindly signs voter credentials. Its

role is similar to that of the bank in an e-coin system.

- *Polling station*: Receives ballots cast by participants and checks their validity. It plays a role similar to that of a merchant. We are assuming that the polling station has been distributed between several servers so as to gain fault tolerance.

- *Key Storage Trusted Party (KSTP)*: This entity publishes the public information needed to encrypt the votes and stores the secret key material required for their decryption. The cryptosystem employed must provide a publicly verifiable decryption.

Protocol steps

We assume the existence of an underlying offline e-coin system which is untraceable and unforgeable. For simplicity, we will refer to the polling station as the server on which the voter is going to cast her ballot. The system defines an election protocol consisting of three steps.

1. *Obtaining a voting credential:* Participants contact the authentication server and request a voting credential. This is done by executing the withdrawal protocol in the underlying e-coin payment system. The authentication server, acting as the bank, first checks that the authenticated participant has not yet received her voting credential. If this is the case, the withdrawal protocol is completed and, as a result, the participant obtains an e-coin. This e-coin is the voting credential that will permit her to cast a vote.

2. *Voting process:* In order to balance the amount of ballots in each polling station server, each participant is assigned one of them. The communications between a participant and the polling station are carried out through an anonymous channel as is usual in blind signature based e-voting. During the voting period, the participant (in the payer role) first composes her vote v, encrypts it (under KSTP's public key), obtaining ciphertext V, and then contacts the polling station (playing the merchant role) through the anonymous channel. The participant sends her voting credential CMT (commitment message of the *payment protocol*). Next, the polling station checks that CMT is a valid credential generated by the authentication server and that it has not been received before. If the voting credential is correct, it returns a random value S (which could be obtained by composing a polling station identifier, a timestamp and some additional random data). After that, the participant computes the challenge $CH = \mathcal{H}(CMT||S||V)$, with $\mathcal{H}$ being some cryptographic hash function like SHA256 [NIS94], and generates the response RSP to CH. The participant sends the

tuple (V, RSP) to the polling station which computes CH by itself (from CMT, S and V) and verifies the response RSP. If all checks are satisfied, it stores the authenticated vote (CMT, V, S, RSP) and publishes it on the public list of votes received. Any entity can check the validity of this tuple, ensuring that only voters with a valid e-coin are allowed to vote.

3. *Tallying:* When the voting period ends, the polling station servers check the list of votes received published by the other servers. If a credential CMT has been used in different polling station servers, the ballots are removed from both lists and the identity of the double voter can be disclosed. After that, the polling station servers ask the KSTP to perform a verifiable decryption of the remaining ballots and publish the resulting cleartexts (the votes) together with the proofs of correct decryption on a publicly accessible bulletin board.

Comments

Our construction makes use of the Fiat-Shamir construction [FS87] by computing the challenge from a hash computation that takes the encrypted vote V as input. In this way, the tuple (CMT, RSP) becomes a digital signature on $S||V$. A random seed S generated by the polling station is included as input, so that a malicious voter casting the same vote V twice will be required to respond to two different challenges.

If a participant is detected to have voted twice (by observing two votes sharing the same value for CMT), her identity will be obtained by means of the anonymity revocation procedure provided by the underlying e-coin system.

5.4.3 Security

The main issue with the previous proposals providing double-voter anonymity revocation was the lack of security. For this reason, it is specially important to prove that the construction presented provides the security requirements described in Section 1.2. One of the main advantages of the scheme is its capacity to run with different e-coin protocols. For this reason, the security of the whole e-voting scheme depends on the security of the underlying e-coin used.

Privacy

This property guarantees that the relation between a vote and the identity of the person who cast it remains unknown. In our protocol, voters cast their ballots through an anonymous channel. Several proposals exist for anonymous channels. In our construction, the anonymous channel is required to

allow bidirectional communications, keeping the identity of the communication establisher secret [RMPAFD14]. Since the votes are received by the polling station through an anonymous channel, the only way to identify the caster of a vote would be to trace the attached e-coin. If the underlying e-coin system is untraceable, this is not possible.

More formally, let ECoin be the offline e-coin system used in the protocol described. Let EVoteCoin be the e-voting system presented in the previous section. Let us assume that $\mathcal{A}$ is an adversary who wants to know the identity of the voter who cast a given ballot received by the polling station. The adversary $\mathcal{A}$ could be a party of voters jointly with a group of external viewers. They are also able to collude with the polling station. Taking that into account, the adversary $\mathcal{A}$ has information on the private information of some voters, and also on the public information generated for the election.

Lemma 5.4.1. *If ECoin is untraceable, EVoteCoin guarantees privacy.*

Proof. We will show that, if an adversary $\mathcal{A}$ were able to identify the voter who cast a given ballot, he could also trace the identity of the spender of a given e-coin for ECoin. Let us assume that EVoteCoin did not provide privacy. That would mean that $\mathcal{A}$ can run an algorithm Alg_1 that is able to find the identity of the voter who cast a given ballot. In order to obtain the identity of the voter who cast a given ballot $(CMT_i, V_i, S_i, CH_i, RSP_i)$, $\mathcal{A}$ would run an algorithm $Alg_1(CMT_i, V_i, S_i, CH_i, RSP_i, I)$, which would return the identity of the voter, where I is all the public information of EVoteCoin, and the private information of the party of voters who constitute $\mathcal{A}$. Under that assumption, $\mathcal{A}$ could use Alg_1 to trace the identity of the spender of an e-coin in ECoin by running the algorithm in this way: $Alg_1(CMT_i, \emptyset, \emptyset, CH_i, RSP_i, I)$, which would return the identity of the spender so that ECoin would be traceable. □

Integrity

An e-voting protocol provides integrity if authentication, unicity and fairness properties are satisfied.

The authentication property requires being able to differentiate between genuine voters and the rest. We use an authentication server which provides a voting credential only to participants listed in the electoral roll. Voters who are not in the electoral roll will not receive an e-coin from the authentication server so they would have to forge one. If our e-coin system is unforgeable and the withdrawal protocol is secure, the only chance for an adversary is to collude with the authentication server. In our protocol, the authentication server is a trusted entity so, as long as our e-coin system is secure against forgery, our voting protocol will not allow votes from people not listed in the electoral roll.

In order to prevent a malicious voter from voting more than once, the authentication server must provide only one voting credential (e-coin) to each voter. A malicious voter could use her e-coin to cast more than one vote, but this situation would be detected (all those votes would share the same CMT) by the polling station, which would proceed accordingly (for instance, revoking that voter's anonymity). Voters could still cast more than one vote if they were able to collude with the authentication server, but as long as authentication server is honest, the only way to do that is by forging e-coins.

Let us assume the adversary $\mathcal{A}$, as defined in Section 5.4.3, tries to cast a valid vote for a participant who does not appear in the electoral roll. This is similar to trying to vote twice. The definition of $\mathcal{A}$ allows the use of private information of several voters so that, in our case, proving that $\mathcal{A}$ cannot vote as a participant not in the electoral roll, is the same as proving that $\mathcal{A}$ cannot cast more than one vote for each voter.

Lemma 5.4.2. *Assuming that the authentication server is honest, EVote-Coin provides integrity as long as ECoin is unforgeable.*

Proof. We will show that an adversary able to cast a vote without being in the electoral roll could also forge a valid e-coin. Let us assume that EVote-Coin did not provide authentication. That is, the adversary $\mathcal{A}$ can run two algorithms Alg_1, Alg_2 that can generate a valid vote without using private information of any voter or of the authentication server. More formally, $Alg_1(I) \rightarrow CMT$, where I is the set of all the available public data together with the private information from $\mathcal{A}$, and CMT is a valid commitment for the e-voting protocol. Besides, $Alg_2(CMT, CH, I) \rightarrow RSP$, where RSP is a valid response for the given CMT and $CH = \mathcal{H}(CMT||V||S)$.

Under that assumption, the adversary could use Alg_1 and Alg_2 to forge a valid coin for ECoin, behaving as follows: The adversary $\mathcal{A}$ would send $Alg_1(I) = CMT$ to the merchant, where I is the public information of the system. The merchant would answer with a challenge CH or some values needed to generate it, and after that $\mathcal{A}$ would run $Alg_2(CMT, CH, I)$ to obtain RSP, which would be a valid response for the payment protocol of ECoin. Hence, ECoin would be forgeable and the claim follows. □

Verifiability

The proposed system provides *individual verifiability*, since a voter can check that her vote has been taken into account by checking that it appears on the public list of votes received. A voter can also check that the other votes are part of a valid (CMT, V, S, RSP) tuple and verify that their decryptions were performed correctly by checking the published decryption proofs.

End-to-end verifiability in blind signature-based voting systems would require the use of a verifiable anonymous channel. However, the efficiency

of the proposal would be severely affected by the verification proofs of such a channel.

Robustness

Credential-based e-voting is more robust than the classical blind signature-based approach. This is because the voters can ask for their credentials before having decided their vote. In this way, credentials can be provided during a long time period before the day of the election, so that there is time to recover from eventual attacks against the authentication server. Moreover, the use of a distributed polling station also increases robustness.

Chapter 6

Conclusions and future work

In this thesis we focus on the study of secure protocols for remote electronic voting. As mentioned throughout this document, the systems for remote electronic voting can be classified into three paradigms: mix-type, homomorphic tallying and blind signature-based. Each of them provides different advantages and drawbacks that have to be studied in order to find an appropriate solution for each election. The proposals presented in this thesis provide novel solutions to each of the existing voting paradigms.

In Chapter 3, the performance issues of the mix-type paradigm are introduced. Based on the fact that these performance issues are caused by a time-consuming zero knowledge proof of a shuffle, we propose a new approach to guarantee the correctness of a shuffle. The resulting scheme is a secure and fully operable mix-type remote voting system belonging to the "proof of product with redundancy" type. This scheme takes advantage of the homomorphic property of the cryptosystem used to validate the equality of the input and output ballots. The system requires the ballots to be properly cast and, for this reason, additional proofs are required during the ballot casting phase. This way, the proposal considerably reduces the time required to obtain the results but increases the interaction between voters and the ballot collection authority.

This proposal was presented in the Electronic Government and the Information Systems Perspective (EGOVIS) conference in 2011 and published in the LNCS proceedings under the title "Verifiable encrypted redundancy for mix-type remote electronic voting" [MMS11].

A novel contribution is also presented for the homomorphic tallying paradigm in Chapter 4. The proposal makes use of a zero-knowledge proof of a shuffle to guarantee the integrity of the election without using range proofs. By avoiding the need to employ range proofs, the resulting system reduces the computational cost required to prove that a ballot has been properly generated. This new hybrid remote voting system bases its efficiency on the shuffle proofs instead of the commonly used range proofs.

The proposal was described in the paper entitled "A hybrid approach to vector-based homomorphic tallying remote voting" [MMS15], published in the International Journal of Information Security in 2015 .

The last paradigm introduced in Chapter 5 is the blind signature-based paradigm. Two contributions to this paradigm are presented. In the first one, a novel blind signature-based remote voting scheme is presented. The system provides anticipated voter authentication and allows a highly distributed (and trusted) authentication server, together with universal verifiability. The use of cryptography over gap Diffie-Hellman groups is studied as a solution which avoids the need for a trusted private key dealer and gives the possibility of speeding up signature validation employing batch techniques.

This proposal was presented in the Information Technology: New Generations (ITNG) conference in 2013 under the title "Blind Certificates for Secure Electronic Voting" [MSV13].

The second proposal focuses on the problem of identifying double voters in the blind signature-based paradigm. We first show that the contribution [BMA+11] to credential-based remote voting is weak. Its security flaw, presented in Section 5.3.5, left that variant of blind signature-based voting as an insecure paradigm. Next, we present a construction that, employing an offline e-coin protocol, builds a secure electronic voting system that follows this variant. The security of our construction holds on the security properties provided by the underlying e-coin system. Basically, the properties required in the e-coin system are unforgeability and untraceability.

This proposal was published in the Journal of Network and Computer Applications in 2014 under the title "Constructing credential-based E-voting systems from offline E-coin protocols" [MSV14].

Electronic elections have a critical impact when used for critical decisions. Even though the aforementioned contributions are secure, in order to prevent attacks caused by malicious software in the voter's computer it would be interesting to study, in the future work, the compatibility of our proposals with well-known cast-as-intended techniques. Moreover, it would be interesting to study the inclusion of Groth-Sahai proofs in [MMS11], so as to increase performance. In addition, an adaptation to provide everlasting privacy in each scheme proposed would be interesting, not only for security purposes but also to analyze the restrictions of everlasting privacy in terms of verifiability and performance.

Bibliography

[AJ09] Mahdi Asadpour and Rasool Jalili. Double voting problem of some anonymous e-voting schemes. *Journal of Information Science and Engineering*, 25(3):895–906, 2009.

[AMPQ09] Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of helios. In *Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE)*, 2009.

[AS08] Hassan Al-Shalabi. E-voting scheme over internet. *Business and Information*, 2008.

[BCG+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. Sok: A comprehensive analysis of game-based ballot privacy definitions. In *IEEE Symposium on Security and Privacy*, pages 499–516. IEEE Computer Society, 2015.

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology - EUROCRYPT*, volume 7237 of *LNCS*, pages 263–280. 2012.

[BGR98] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology – EUROCRYPT*, volume 1403 of *LNCS*, pages 236–250. 1998.

[BMA+11] Yaser Baseri, Amir S Mortazavi, Maryam Rajabzadeh Asaar, Mohsen Pourpouneh, and Javad Mohajeri. Double voter perceptible blind signature based electronic voting protocol. *The ISC International Journal of Information Security*, 3(1):43–50, 2011.

[Bra94] Stefan Brands. Untraceable off-line cash in wallet with observers. In *Advances in Cryptology – CRYPTO*, volume 773 of *LNCS*, pages 302–318. 1994.

[CCM08] Michael Clarkson, Stephen Chong, and Andrew Myers. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368, 2008.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology – CRYPTO*, volume 839 of *LNCS*, pages 174–187. 1994.

[CEC+08] David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi Vora. Scantegrity: End-to-end voter-verifiable optical- scan voting. *Security Privacy, IEEE*, 6(3):40–46, 2008.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[Cha88] David Chaum. Elections with unconditionally-secret ballots and disruption equivalent to breaking rsa. In *Advances in Cryptology – EUROCRYPT*, volume 330 of *LNCS*, pages 177–182. 1988.

[CJT03] Hung-Yu Chien, Jinn-Ke Jan, and Yuh-Min Tseng. Cryptanalysis on Mu–Varadharajans e-voting schemes. *Applied Mathematics and Computation*, 139(2):525–530, 2003.

[CLW08] Sherman S. M. Chow, Joseph K. Liu, and Duncan S. Wong. Robust receipt-free election system with ballot secrecy and verifiability. In *NDSS*, volume 8, pages 81–94, 2008.

[CP93] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Advances in Cryptology – CRYPTO*, volume 740 of *LNCS*, pages 89–105. 1993.

[DDS09] George Danezis, Claudia Diaz, and Paul Syverson. Systems for anonymous communication. *Handbook of Financial Cryptography and Security, Cryptography and Network Security Series*, pages 341–389, 2009.

[DH76] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[dHSV09] Sebastiaan de Hoogh, Berry Schoenmakers, Boris kori, and José Villegas. Verifiable rotation of homomorphic encryptions. In *Public Key Cryptography – PKC*, volume 5443 of *LNCS*, pages 393–410. 2009.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: PKC*, volume 1992 of *LNCS*, pages 119–136. 2001.

[DK00] Yvo Desmedt and Kaoru Kurosawa. How to break a practical mix and design a new one. In *Advances in Cryptology – EUROCRYPT*, volume 1807 of *LNCS*, pages 557–572. 2000.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FG07] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, 2007:15, 2007.

[FOO93] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Advances in cryptology – AUSCRYPT*, volume 718 of *LNCS*, pages 244–251. 1993.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO*, volume 263 of *LNCS*, pages 186–194. 1987.

[FS01] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *LNCS*, pages 368–387. 2001.

[GL07] Jens Groth and Steve Lu. Verifiable shuffle of large size ciphertexts. In *Public Key Cryptography (PKC)*, volume 4450 of *LNCS*, pages 377–392. 2007.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[Gro05] Jens Groth. Non-interactive zero-knowledge arguments for voting. In *Applied Cryptography and Network Security*, volume 3531 of *LNCS*, pages 467–482. 2005.

[GZB+02] Philippe Golle, Sheng Zhong, Dan Boneh, Markus Jakobsson, and Ari Juels. Optimistic mixing for exit-polls. In *Advances in Cryptology – ASIACRYPT*, volume 2501 of *LNCS*, pages 451–465. 2002.

[HMP95] Patrick Horster, Markus Michels, and Holger Petersen. Blind multisignature schemes and their relevance to electronic voting. In *11th Anual Computer Security Applications Conference*, pages 149–155, 1995.

[HS98] Qi He and Zhongmin Su. A new practical secure e-voting scheme. In *14th Intl. Information Security Conference*, pages 196–205, 1998.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology – EUROCRYPT*, volume 1807 of *LNCS*, pages 539–556. 2000.

[HWH05] Sheng-Yu Hwang, Hsiang-An Wen, and Tzonelih Hwang. On the security enhancement for anonymous secure e-voting over computer network. *Computer Standards & Interfaces*, 27(2):163–168, 2005.

[Jak98] Markus Jakobsson. A practical mix. In *Advances in Cryptology – EUROCRYPT*, volume 1403 of *LNCS*, pages 448–461. 1998.

[JCPACRV12] Roger Jardí-Cedó, Jordi Pujol-Ahulló, Jordi Castella-Roca, and Alexandre Viejo. Study on poll-site voting and verification systems. *Computers & Security*, 31(8):989–1010, 2012.

[JJ01] Markus Jakobsson and Ari Juels. An optimally robust hybrid mix network. In *Procs. of the 20th annual ACM symposium on Principles of distributed computing*, pages 284–292. ACM Press, 2001.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium*, pages 339–353. USENIX Association, 2002.

[JLL02] Wen-Shenq Juang, Chin-Laung Lei, and Horng-Twu Liaw. A verifiable multi-authority secret election allowing abstention from voting. *Comput. J.*, 45(6):672–682, 2002.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[KW13] Shahram Khazaei and Douglas Wikström. Randomized partial checking revisited. In *Topics in Cryptology CT-RSA*, volume 7779 of *LNCS*, pages 115–128. 2013.

[LHC03] Iuon-Chang Lin, Min-Shiang Hwang, and Chin-Chen Chang. Security enhancement for anonymous secure e-voting over a network. *Computer Standards & Interfaces*, 25(2):131–139, 2003.

[LK03] Byoungcheon Lee and Kwangjo Kim. Receipt-free electronic voting scheme with a tamper-resistant randomizer. In *Information Security and Cryptology – ICISC*, volume 2587 of *LNCS*, pages 389–406. 2003.

[LLCC06] Jian-Liang Lin, Hsiu-Feng Lin, Chih-Ying Chen, and Chin-Chen Chang. A multiauthority electronic voting protocol based upon a blind multisignature scheme. *International Journal of Computer Science and Network Security*, 6(12):266–274, 2006.

[LTW05] Joseph K. Liu, Patrick P. Tsang, and Duncan S. Wong. Recoverable and untraceable e-cash. In *Public Key Infrastructure (PKI)*, volume 3545 of *LNCS*, pages 206–214. 2005.

[Mil86] Victor Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology – CRYPTO*, volume 218 of *LNCS*, pages 417–426. 1986.

[ML98] Wenbo Mao and Chae Hoon Lim. Cryptanalysis in prime order subgroups of $\mathbb{Z}_n^*$. In *Advances in Cryptology – ASIACRYPT*, volume 1514 of *LNCS*, pages 214–226. 1998.

[MMS11] Víctor Mateu, Josep M. Miret, and Francesc Sebé. Verifiable encrypted redundancy for mix-type remote electronic voting. In *Electronic Government and the Information Systems Perspective*, volume 6866 of *LNCS*, pages 370–385. 2011.

[MMS15] Víctor Mateu, Josep M. Miret, and Francesc Sebé. A hybrid approach to vector-based homomorphic tallying remote voting. *International Journal of Information Security*, pages 1–11, doi:10.1007/s10207-015-0279-8, in press, 2015.

[MS11] Josep M Miret and Francesc Sebé. Cryptanalysis of an ad-hoc cryptosystem for mix-based e-voting robust against relation attacks. *International Journal of Information Security*, 10(6):387–389, 2011.

[MSV13] Víctor Mateu, Francesc Sebé, and Magda Valls. Blind certificates for secure electronic voting. In *Information Technology: New Generations (ITNG)*, pages 20–26. IEEE Press, 2013.

[MSV14] Víctor Mateu, Francesc Sebé, and Magda Valls. Constructing credential-based e-voting systems from offline e-coin protocols. *Journal of Network and Computer Applications*, 42:39–44, 2014.

[MV98] Yi Mu and Vijay Varadharajan. Anonymous secure e-voting over a network. In *14th Annual Computer Security Applications Conference (ACSAC)*, pages 293–299. IEEE Computer Society, 1998.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS)*, pages 116–125. ACM, 2001.

[NIS94] NIST. Standard, secure hash. *Federal Information Processing Standard*, 180-2:57–71, 1994.

[OMA+99] Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka, and Tatsuaki Okamoto. An improvement on a practical secret voting scheme. In *Information Security Workshop (ISW)*, volume 1729 of *LNCS*, pages 225–234. 1999.

[PAB+04] Kun Peng, Riza Aditya, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Multiplicative homomorphic e-voting. In *Progress in Cryptology – INDOCRYPT*, volume 3348 of *LNCS*, pages 61–72. 2004.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT*, volume 1592 of *LNCS*, pages 223–238. 1999.

[PB09] Kun Peng and Feng Bao. Efficient vote validity check in homomorphic electronic voting. In *Information Security and Cryptology – ICISC 2008*, volume 5461 of *LNCS*, pages 202–217. 2009.

[PB11] Kun Peng and Feng Bao. Efficient multiplicative homomorphic e-voting. In *Information Security Conference – ISC 2010*, volume 6531 of *LNCS*, pages 381–393. 2011.

[Pen11a] Kun Peng. An efficient shuffling based evoting scheme. *Journal of Systems and Software*, 84(6):906–922, 2011.

[Pen11b] Kun Peng. A general and efficient countermeasure to relation attacks in mix-based e-voting. *International Journal of Information Security*, 10(1):49–60, 2011.

[Pfi95] Birgit Pfitzmann. Breaking efficient anonymous channel. In *Advances in Cryptology – EUROCRYPT 1994*, volume 950 of *LNCS*, pages 332–340. 1995.

[PG10] Jordi Puiggalí and Sandra Guasch. Universally verifiable efficient re-encryption mixnet. In *Electronic Voting 2010, EVOTE, 4th International Conference*, volume 167 of *LNI*, pages 241–254. GI, 2010.

[PH78] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24(1):106–110, 1978.

[PIK94] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology – EUROCRYPT 93*, volume 765 of *LNCS*, pages 248–259. 1994.

[PP90] Birgit Pfitzmann and Andreas Pfitzmann. How to break the direct rsa-implementation of mixes. In *Advances in Cryptology – EUROCRYPT*, volume 434 of *LNCS*, pages 373–381. 1990.

[Rad95] Michael J. Radwin. An untraceable, universally verifiable voting scheme. In *Seminar in Cryptology*, 1995.

[RHOAGZ07] Francisco Rodríguez-Henríquez, Daniel Ortiz-Arroyo, and Claudia García-Zamora. Yet another improvement over the Mu–Varadharajan e-voting protocol. *Computer Standards & Interfaces*, 29(4):471–480, 2007.

[RMPAFD14] David Rebollo-Monedero, Javier Parra-Arnau, Jordi Forné, and Claudia Diaz. Optimizing the design parameters of threshold pool mixes for anonymity and delay. *Computer Networks*, 67:180–200, 2014.

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

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme. In *Advances in Cryptology – EUROCRYPT 95*, volume 921 of *LNCS*, pages 393–403. 1995.

[SMPP10] Francesc Sebé, Josep M. Miret, Jordi Pujolàs, and Jordi Puiggalí. Simple and efficient hash-based verifiable mixing for remote electronic voting. *Computer Communications*, 33(6):667–675, 2010.

[SS98] Joseph H. Silverman and Joe Suzuki. Elliptic curve discrete logarithms and the index calculus. In *Advances in Cryptology – ASIACRYPT*, volume 1514 of *LNCS*, pages 110–125. 1998.

[TW10] Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In *Progress in Cryptology – AFRICACRYPT*, volume 6055 of *LNCS*, pages 100–113. 2010.

[Wik04] Douglas Wikström. Five practical attacks for optimistic mixing for exit-polls. In *Selected Areas in Cryptography*, volume 3006 of *LNCS*, pages 160–174. 2004.

[Wik09] Douglas Wikström. A commitment-consistent proof of a shuffle. In *Proceedings of the 14th Australasian Conference on Information Security and Privacy*, volume 5594 of *LNCS*, pages 407–421. 2009.

[YLY04] Chou-Chen Yang, Ching-Ying Lin, and Hung-Wen Yang. Improved anonymous secure e-voting over a network. *Information & Security*, 15(2):185–191, 2004.