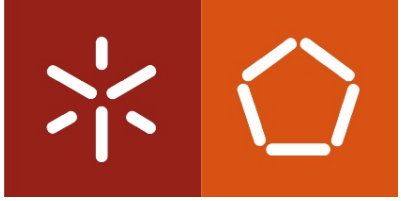
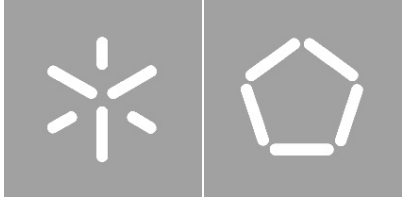


Universidade do Minho
Escola de Engenharia

Pedro Vasconcelos Castro Lopes Faria

Remote Electronic Voting
Studying and Improving Helios

Dezembro de 2012



Universidade do Minho
Escola de Engenharia

Pedro Vasconcelos Castro Lopes Faria

Remote Electronic Voting

Tese de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Doutor José Carlos Bacelar Almeida

Dezembro de 2012

Remote Electronic Voting

Studying and Improving Helios

Pedro Vasconcelos Castro Lopes Faria

(pg17684@alunos.uminho.pt)

Dissertation submitted in partial fulfillment of the requirements for the degree of Master in Informatics Engineering at the University of Minho, under the supervision of

José Carlos Bacelar Almeida

Departamento de Informática
Escola de Engenharia
Universidade do Minho

Braga, December 30, 2012

Abstract

A former North American President once said that *the ballot is stronger than the bullet*. In fact, the most civilized and organized way for a people express their opinion is by voting. However, there are people with bad intentions that affect voting and elections, being normal situations of coercion, collusion, fraud or forgery that disturb and cause alterations in the outcome of a vote. Thus, it becomes necessary to find ways to protect the voters, through vote secrecy and transparency, so that in end of a voting, democracy and justice prevail. Since the secret ballot papers until the electronic voting machines, passing through punched cards, technology in voting systems is evolving to ensure a greater security in elections, as well as greater efficiency, lower costs and other characteristics wanted in this type of systems. Nowadays, remote electronic voting is seen as the ultimate goal to achieve. The difficulty of developing such system is to ensure that it meets all the security requirements without infringing each other and without compromising the usability of the system itself. Thus, cryptography becomes an essential tool for obtaining security and integrity on electronic voting systems.

This master thesis focuses on the world of electronic voting, in particular, the remote electronic voting. The objective is to find a system of this kind, with real world applications, to be studied and analyzed in a security point of view. Hence, we made a research on voting and, more deeply, a research on electronic voting schemes, in order to learn how to conceive it, which include the different stages that compose an election, types of voting and the entities involved, and what requirements to fulfill, both the security and functional. Because cryptography is used in most schemes, a detailed study was also performed on the primitives most common in protocols of electronic voting. However, there are not many schemes that pass from theory to practice. Fortunately, we found Helios, a well known scheme that implements various cryptographic techniques for everyone, under certain assumptions, be able to audit polls conducted with this system. A study was performed in order to explain how it was constructed and to identify its strengths and weaknesses. We also present some ongoing work by different people to improve Helios. Finally, we propose improvements on our own, to fight against coercion, to decrease the levels of assumptions and overcome corruption issues. Furthermore, we propose measures to protect the virtual voting booth and a mobile application to cast votes.

Resumo

Um antigo Presidente norte americano disse um dia que *o voto é mais forte que a bala*. De facto, a forma mais civilizada e organizada de um povo exprimir as suas opiniões é através de votações. Infelizmente, também este mundo é afectado por pessoas com más intenções, sendo normais as situações de coação, conluio, fraude ou falsificação que perturbam e causam alterações no resultado de uma votação. Assim, torna-se necessário arranjar formas de proteger os votantes, através de segredo de voto e transparência, de forma que, no final, a democracia e justiça de uma votação prevaleçam. Desde dos boletins de papel secreto até às máquinas de voto electrónico, passando pelas *punched cards*, a tecnologia em sistemas de votação vem evoluindo de modo a garantir uma maior segurança em eleições, assim como maior eficiência, menor custos e outras características que se querem neste tipo de sistemas. Nos dias de hoje, o voto electrónico remoto é visto como o grande objectivo a cumprir. A grande dificuldade de se desenvolver tal sistema é garantir que o sistema cumpra todos os requisitos de segurança sem que se violem entre si e sem que isso prejudique a usabilidade do sistema em si. Assim, a criptografia torna-se uma ferramenta essencial para se obter segurança e integridade em sistemas de voto electrónico.

Esta tese de mestrado foca-se no mundo do voto electrónico, mais especificamente o voto electrónico remoto. O grande objectivo seria arranjar um sistema desse tipo, que tivesse aplicação real, para ser estudado e analisado do ponto de vista de segurança. Fez-se então uma pesquisa necessária sobre votações e, mais aprofundada, uma sobre esquemas de voto electrónico, de modo a aprender como se concebem, tanto as fases que a constituem como as entidades que normalmente fazem parte, e quais os requisitos a cumprir, tanto os funcionais como os de segurança. Como a criptografia entra em grande parte dos esquemas, também um estudo aprofundado foi realizado sobre as primitivas mais comuns em protocolos de voto electrónico. No entanto, não existem muitos esquemas que passem da teoria à prática. Felizmente, encontrou-se o Helios, um sistema que põe em prática diversas técnicas criptográficas para que qualquer pessoa, dentro de certas assumpções, possa auditar votações conduzidas por este sistema, ficando a privacidade nas mãos do Helios. Um estudo foi realizado de modo a explicar como foi construído e identificar os seus pontos fortes e fracos. Também são apresentados alguns trabalhos em curso sobre este sistema. Finalmente, propõem-se outros tipos de melhoramentos que visam: combater coação, diminuir o nível das assumpções e ultrapassar problemas de corrupção. Propõem-se ainda medidas para proteger a cabine virtual de votação e uma aplicação móvel.

Acknowledgments

First of all, I want to thank Professor José Carlos Bacelar Almeida for the overall support, the research guidance and also for all the suggestions that improved this dissertation. I am also grateful to: Professor Manuel Barbosa, for suggesting Helios as a case study; Professor José Valença, for interesting discussions about e-Voting; and to Gonçalo Hermenegildo from MULTICERT, for giving me an insight about voting in Portugal.

To all my friends, roommates and Confraria do Matador, thank you for your support and positive thinking. I also want to thank the SSaaPP project members for their understanding and patience during the last stage of this work. A special thanks to Leonel, I am very grateful for all the help you provided and for the endless discussions we had about security.

I wish to thank Vânia. You always kept me happy during the darkest hours.

And finally, I don't have the words to describe how much thankful I am to my parents and sister. Thank you for everything.

Preface

This document is a master thesis in Informatics Engineering (area of Cryptography and Security) submitted to Universidade do Minho, Braga, Portugal.

Notes to the reader:

- All links were checked at end of writing this document - 30 December.
- Throughout the document the academic plural appears often in the text to describe the work developed. This form was intentionally used for two reasons: first, some of the work here presented was done in cooperation; and second, the plural can help the reader to feel more closely connected with the work done.
- Some images may be hard to understand. In the electronic version of this document, they are links if one wishes to download and see them in full size.

Contents

List of Figures

List of Examples

Acronyms

API	Application Programming Interface
BB	Bulletin Board
BS	Blind Signature
CA	Certificate Authority
DE	Deniable Encryption
DoS	Denial-of-Service
DDH	Decision Diffie-Hellman
DLP	Discrete Logarithm Problem
DRE	Direct-recording Electronic
DS	Digital Signature
E2E	End-to-End
e-voting	Electronic Voting
EEG	Exponential ElGamal Encryption
ElGamal	ElGamal Encryption
HTML	HyperText Markup Language
HE	Homomorphic Encryption
i-voting	Internet Voting
IACR	International Association for Cryptologic Research
JS	JavaScript
JSON	JavaScript Object Notation

mixnet Mixed Network

PKC Public-key Cryptography

RA Registration Authority

RSA Rivest, Shamir and Adleman

SBT Smart Ballot Tracker

SHA Secure Hash Algorithm

SS Secrete Sharing

URL Uniform Resource Locator

ZKP Zero-Knowledge Proof

ZKS Zero-Knowledge Sets

Chapter 1

Introduction

The first lesson is this: take it from me, every vote counts.

Al Gore

The above quote came four years after one the biggest scandals regarding elections. It was the year 2000, and it opposed Al Gore against George W. Bush for the position of the President of the United States[1]. What was later known as the Florida recount, started with different television companies announcing different winners before the polling was over. In a close race, in which the winner won by less than 500 from 6.000.000 votes, there was a lot of controversy involved in the tallying stage. Even a few years later, some independent studies continue to give different winners for that election. But, leaving aside conspiracy theories, some problems were actually found involving voting technologies. The biggest was on the usability and design of ballots[2], where using the called butterfly ballots resulted in casting wrong votes. This and other problems, urge the need to make reforms on the voting system of that country.

More recently, last year in Russia a wave of protest arose all over the country. It started precisely with the 2011 Russian legislative elections , where a large number of people, under the slogan "For Fair Elections", contested the election results[3] by claiming that most of the major problems that affect

[1] Complete story of the presidential election and scandal: http://en.wikipedia.org/wiki/United_States_presidential_election,_2000

[2] (In Portuguese) The problem: http://www.publico.pt/Cultura/quando-o-design-corre-mal-1559435.

[3] 2011-2012 Russian Protests: http://en.wikipedia.org/wiki/2011%E2%80%932012_Russian_protests.

elections had happened[4], such as:

- Election fraud and falsification of several partial election results;
- Obstruction of observers;
- Illegal campaigning;
- Multiple voting;
- Coercion with threats of violence;

But even small organizations, cities or villages are susceptible to corruption in elections. For instance, currently, more than one fifth of the surveyed voters in Açores are "ghost-voters", i.e. dead people with permission to vote[5]. If this at first sight may not seem like a big deal, one may think that the extra votes would be simply discarded, given corruption it may lead to other severe problems. For instance, one may use the extra votes for vote buying or ballot stuffing.

There are many more examples of fraud, corruption, mistakes or deceptions regarding real-world elections. To overcome these problems, new voting types, voting systems and new technologies are always emerging in order to create truly fair and transparent elections. But new solutions promote new ways of corruption, so they are always temporary and subject to improvement. For instance, regarding voting technologies, if lever machines and punchcards are history, the Direct-recording Electronic (DRE) voting machine, was already introduced over 20 years ago and is still suffering changes. In turn, although it was the first to appear, paper ballots have not yet loose importance in voting systems, being normally the first choice when creating elections. Even so, some countries have already established elections that ran exclusively on Electronic Voting (*e*-voting) machines, being Brazil the first country to run government elections solely in DRE machines. Estonia in turn, is seen as the most technologically advanced country when the matter is voting systems. In spite the lack of security and the still not achieved security requirements, they already conducted online elections, a kind of remote *e*-voting, which by now is seen as the greatest and last goal of voting systems. Unfortunately, some countries are still far from conducting such elections, due to financial difficulties, high standards for security requirements or even legal issues. For

[4]Guardian news: http://www.guardian.co.uk/global/2011/dec/10/russia-elections-putin-protest.

[5]The complete story: http://www.publico.pt/Pol%C3%ADtica/mais-de-um-quinto-dos-recenseados-nos-acores-sao-eleitoresfantasma-1566193.

example, in the Constitution of the Portuguese Republic, the 121^{th} article states: the right to vote in national territory is exercised in person[6].

1.1 Motivation

Online elections is already a service wide explored in the market. From a quick research, one is able to find several companies, such as SimplyVoting, Election Buddy or Votenet[7], that provide solutions whose goal is to achieve the main requirements of elections: secrecy and democracy. But since "in business, silence is golden", most companies, to protect their business, fall into the mistake of making end-users to blindly trust that their votes are not tampered or their privacy violated. Such conclusion is pointed out in [Gerck et al., 2002], where they also claim that *e*-voting is not like any other electronic commerce, since they meet different criteria. For instance, a receipt from an electronic transaction cannot be like a receipt from a casted vote, since it could lead to severe problems.

Thus, there is a need to create systems that can be fully audited by anyone who uses it. Even for security reasons, is not good to rely on obscurity. Using an interesting analogy of [Schneier, 1996], taking a letter, locking it in a safe, and hide it somewhere in the world for one to crack it is not achieving security. On the other hand, provide documentation of the safe, its "source code" and all kind of specifications and one still not be able to crack it, that is achieving security. The same principles should be followed when designing voting systems. Here, cryptography plays an important role, providing the tools necessary to create protocols where the voting requirements are fulfilled, and each end-user will be able to verify it.

Nevertheless, there is always a need of an assumption of trust in order to prove the security of an *e*-voting scheme. The problem with most of online voting services is the high level assumption of trusting in the system for all integrity and privacy matters related to elections. In literature, is common to find authors that express concern about it, and not only in *e*-voting schemes. One practical solution is the inclusion of external entities on the designs to split or share responsibilities of election's roles, and can be seen in famous *e*-voting schemes, such as [Fujioka et al., 1993, Benaloh and Tuinstra, 1994]. Thus, many schemes may guarantee its safety without having too strong as-

[6] (In Portuguese) Constitution: http://www.fd.uc.pt/CI/CEE/OI/Constituicao_Portuguesa.htm.

[7] Company's websites: http://www.simplyvoting.com/ http://electionbuddy.com/ http://www.votenet.com/

sumptions.

Finally, there is a special problem regarding remote electronic elections, more concretely Internet Voting (*i*-voting) that uses personal computers as voting booths, which is the one of the main reasons why the remote *e*-voting hypothesis seems so unreal among renown experts, or that it will take years, or even decades, to achieve in a large scale [Gerck et al., 2002]. The problem is that since personal computers are involved, there is no way to guarantee that they are not tampered, with no virus, worms, spyware or any kind of malware. There are solutions, as the use of Live CDs, but since there is no protected environment, one is only capable of postponing the problem. Nevertheless, we believe that if the postponement is great enough, it is justifiable to create and apply security techniques and add them to voting systems.

1.2 Objectives

The main objective of this work is to create, or improve, measures that assure security on a remote *e*-voting system. With that in mind, the path to reach that goal will consist in:

- Entering deeply on the *e*-voting world. Understand the usual requirements and the security properties of general voting systems. Understand also the workflow of an election and learn both the entities and the voting types involved.

- Studying the cryptographic primitives that are usually used to improve security on *e*-voting systems. For each item found, see how it is applied in a scheme and the advantages that it brings.

- Researching remote *e*-voting systems, more concretely *i*-voting systems. From the available systems found, choose one based on the documentation, on the tests performed, on its characteristics and cryptographic techniques involved. Also, the chosen system must open-source, in order to work on it and perform changes if necessary.

- Inventing new ways to improve or modify the security on the chosen system in order to overcome the limitations, faults or fails encoutered on that system.

1.3 Contributions

This work presents an overview of the *e*-voting world, with special attention to the conceptual perspective of an election and the voting requirements. Having a good understanding of how an election works would be an essential and necessary step for evaluating and making changes on a system. Therefore, the conceptual perspective covers the different kind of voting types, the main stages of an election and the entities involved.

This work also presents the most used primitives when creating *e*-voting protocols. So for each one of three main primitives, an informal explanation, with the help of simple analogies, was given, a well cited scheme was chosen to demonstrate its utility and a technique that applies it was described. For the rest of the primitives, a detailed description was presented. Nevertheless, they are of crucial importance to understand the rest of the work.

Helios was the chosen system to work on, due to its characteristics, easy of use and the fact of being open-source. Based on the documentation, source code, papers documenting attacks and uses, thesis, and numerous tests, we present a description of the workflow and techniques used in this system. We also present several ongoing works that aim to improve security on Helios, which include decreasing the level of assumptions of this system and overcome encountered flaws. Finally, we try to improve the system on our own, by:

- *Improving coercion-resistance in Helios*: based on other schemes, we choose to modify the system to have two kind of passwords for the voter to use in different kind of situations. This do not solve the problem, but may increase greatly the protection of the voter. Unfortunately, this also leaves some open problems, specially the discussion of whether is more relevant to have universal verifiability or coercion-resistance.

- *Adding an extra entity to the system*: using the approach of divide to conquer, used in many schemes, this idea occur to solve the biggest assumption of Helios, "Trust Helios for privacy", and a small contradiction detected, when using private elections, if using Helios alone, the access to the virtual booth is not anonymous.

- *Resolving a particular corruption problem*: regarding the registration stage, we design a solution based on a Zero-Knowledge Proof (ZKP), to help to prevent fake voters from entering in an election.

- *Improving authenticity and integrity on virtual booths*: add a feature whose goal is guaranteeing that the booth in which the voter is casting

his vote is authentic. The solution was conceived with code signing.

- *Improving mobility*: A prototype on an Android application for Helios was developed. The goal of this tool was to provide a native application that allow voting from a smartphone without recurring to the phone's slow browser.

1.4 Roadmap

This work is divided into 6 Chapters. The first chapter, the current one, presents an introduction to all the work undertaken for this dissertation.

Chapter 2 introduces the *e*-voting world. It starts by describing the advantages of *e*-voting systems. Then, explains how an election works by presenting the most used voting types, the entities that are usually found in an *e*-voting system and a detailed stages of an electronic election. This chapter ends with a list of security properties and general requirements of this kind of systems.

Chapter 3 presents the cryptographic primitives most used on voting schemes. It gives emphasis to the three most used: Homomorphic Encryption (HE), Blind Signature (BS) and Mixed Network (mixnet). For each one of these primitives, it describes one well known scheme that use the primitive and a detailed technique that implements it. The chapter also presents four other primitives: ZKP, Deniable Encryption (DE), Re-Encryption and Secrete Sharing (SS).

Chapter 4 first starts by presenting a contest for *e*-voting systems and describes the contestants. Then, this chapter analyses a remote *e*-voting system called Helios. By using an example, it describes how an election works and what is possible to do. Next, the backstage of Helios is explained, mentioning the techniques involved. Finally, some security assumptions and current works on this system are presented.

Chapter 5 is divided in five sections. The first explains how a new measure to improve coercion-resistance was created. The second introduces a new entity to the system that will provide a significant decrease on the biggest assumption of Helios. The third deals with a registration problem and the corruption that may derive from it. The fourth with lack of trust in a virtual booth and how to guarantee it. Finally, the last section presents a prototype of an Android application.

In the last chapter we present the conclusion of this dissertation, summarizing the developed work and describing what can be done in the future based on it.

Chapter 2

Electronic Voting

At the bottom of all the tributes paid to democracy is the little man walking into the little booth with a little pencil, making a little cross on a little bit of paper.

Sir Winston Churchill, 1944

Democracy, like many other concepts, has its origin in ancient Greece. It is usually described as a type of political regime in which the power to rule and make decisions belongs to "the people". There are two forms of democracy: *direct democracy*, in which people express their will in each particular subject; and the most common form of democracy, the so-called *representative democracy*, in which they express their will by electing a set of representatives, which in turn make the decisions for them. But both forms of democracy have one crucial point in common: they depend on processes of choice made by the people. This process is generally performed by voting.

Thus, voting can be described as a decision process, in which voters, *i.e.* those previously designated to participate in the voting process, express their opinion through a predetermined choice. The vote then becomes an important process to the preservation of democracy. However, to avoid pressure on voters and coercion by others, it is necessary that the vote is secret. Therefore, there is a need to create mechanisms and schemes that result in systems that ensure the privacy of voters in order to increase fairness in elections, and thus preserve democracy.

In this chapter, an overview about voting, with more emphasis in Electronic Voting (*e*-voting), will be presented. It will start with a brief analysis of pros and cons of using *e*-voting systems in Section 2.1. In Section 2.2, a conceptual

perspective of a general voting system will be given, referring various types of voting, a generic model of a voting system and some common entities involved in it. Section 2.3 will be divided into two subsections, both talking about voting requirements: one to security properties and the other to general requirements. Finally, section 2.4 will summarize this chapter. A small extra section about the history of voting technologies is given in Appendix A.1.

2.1 Pros and Cons of *e-voting*

Although the discussion of pros and cons of *e*-voting may lead to an extensive debate, this section will only briefly summarize points of advantages and disadvantages that can be easily found in the literature.

Preparation. Starting with the preparation of an election, *e*-voting may require a big initial investment for equipment, cost of planning and testing phases. On the other hand, classical paper ballots are easier to understand, do not require complex testing and equipment is far more simple. But this only regards one first election. For further elections, *e*-voting will not require the initial investment, since the equipment already exists, the planning may be adapted and the testing may have already been done. In turn, classical paper ballot must need new equipment, at least for ballots, and has higher costs of manpower. Moreover, in *e*-voting is possible to make last minute changes, for example changing the layout of the ballot, and that may not imply having extra costs.

Voting process. Regarding the voting process, *e*-voting also offers several advantages. The first is the higher level of usability when comparing with paper ballots. It may prevent the casting invalid ballots, since that ballot may be evaluated when filling it. With electronic ballots, ambiguity may be eliminated, since the ballot may contain far more information than the paper version. Concerning people with disabilities or difficulties, electronic ballots can be easily modified to improve the usability according to the person's disability, for instance audio ballot for blind people or extra languages for foreign voters. Another advantage is that *e*-voting will probably speed up the voting process, e.g no waiting lines for submitting a ballot. An important feature of *e*-voting is the possibility of re-voting, i.e. modify the first casted vote, which in the classical paper ballots is not possible. Finally, voting can be made from any poll station, and ultimately, can even be made in home.

Vote counting and results. In several *e*-voting systems is possible to protect the integrity of an election under some assumptions, which on the other kind of voting systems, the voter must blindly trust on the election's results. However, *e*-voting systems are technically more evolved. Hence, for the majority of the voters the whole election's process may by a huge black box. In other words, most people will not understand what is happening and why they can trust in the system. Fraud and tampered results are harder to obtain with *e*-voting systems. Mainly because modifying a digital ballot is not so trivial when comparing with its paper version. Digital ballots are also easily stored and can be audit in a later stage of the election. Recounting is another pro argument, since it is an easy process with no high costs attached. On the other hand, if it is harder to corrupt elections, it is also harder to find corruption. Tampering ballots can be made is such a low level that it makes almost impossible to track corrupted ballots to auditors. When using paper ballots, the entire process can be seen, from the moment that the voter cast a vote, to the transportation of the ballot box and tallying. Even so, *e*-voting brings more benefits, for instance the time for tallying is way more faster when comparing with paper ballots, and the results far more accurate, because errors are more likely to happen when the count is made by humans than by machines.

Other con arguments. Since software is involved with electronic machines, there is always the chance of a particular voting booth be tampered. The machines are prone to hacking and any kind of malware, such as worms, virus, *trojan* or *phishing* attacks. Finally, an error in a centralized system may affect and compromise an entire election.

2.2 Conceptual Perspective

As much as a voting system uses the latest developments/resources regarding web servers, cryptographic techniques, everything related to security, it will never be safe if it does not have a strong security model behind it, i.e. if the model is inconsistent and the tools are not correctly used it becomes impossible to guarantee the safety of the system. Thus, it is necessary to define a model that pay attention to every entity and requirement of the system.

In this section, we will introduce the reader to different types of polls, to a generic description of a *e*-voting system and to a set of the normally used entities and a description of their roles.

2.2.1 Voting and Ballot Types

When modelling a voting system, the type of voting may influence the architecture of the system and technical characteristics, such as security or data storage. Therefore, it is essential to consider any kind of voting to have a well prepared system. In this subsection, several of the most used voting types are introduced.

Single candidate selection This is the most common type of voting, and can be mostly found in government elections. From a set of two or more candidates, the voter can choose at most one, or he can leave his vote blank. In the end, wins the candidate with a higher number of votes. Example 2.1 simulates a ballot.

Candidate	Vote
Alice	
Bob	
Eve	✓

Example 2.1: Single candidate ballot example

Multiple candidate selection A bit different from the previous case. Now the voter, from a set of candidates, has a minimum and a maximum of possible selections. Like the previous example, wins the candidate with a higher number of votes. Examples can be viewed in 2.1 and 2.2.

Candidate	Vote
Alice	✓
Bob	
Eve	✓

Example 2.2: Multiple candidate ballot example

Weighted multiple candidate selection Given an $\mathcal{X}$ number of points, a voter may distribute them in a list of candidates. It may have some rules associated, such for example, the voter can not give all the points to a single candidate. With its popularity increasing, a example of its use in the real world can be seen popular show *Eurovision Song Contest*, where a country award a set of points from 1 to 8, then 10 and finally 12 to other songs in the competition. An illustration of the vote is simulated in 2.3.

Country	Votes
Albania	
Austria	3
Belgium	
...	
United Kingdom	5
Ukraine	10

Example 2.3: *Eurovision* ballot example

Vote changing (a.k.a. Re-Voting) More a property than a type of voting, *Vote changing* elections allow a voter to change his vote until the deadline, where it is casted. This kind of voting usually increases the security complexity of a voting system.

Referendum Commonly used in political affairs, where the people is called to directly decide on a problem, a referendum is a particular kind of voting. It may be seen as a "yes or no" question, just like in Example 2.4, where the voter can even left his vote in blank, if he can not decide.

Question	Yes	No
Should the driving age be lowered to 16?	✓	
Should the drinking age be lowered to 12?		✓

Example 2.4: *Referendum* ballot example

Write-in A *Write-in* vote is a kind of open vote. The voter answers to a decision writing his choice in a ballot. Although it is useful and necessary in some situations, for instance when the voter's choice is not on the set of the candidates and there is a field called "Other" where the voter can write his choice, these kind of votes are usually hard to cast and evaluate. Example of a *Write-in* ballot in 2.5.

Candidate	Vote
Alice	
Bob	
Eve	
Other	*Charlie*

Example 2.5: *Write-in* ballot example

Runoff voting Generally, some elections require that the winner must be elected with more than 50% of the casted votes. As a result, the concept of a *Runoff voting* was born. This kind of election is a continuous process, where after each round, the weakest candidate is eliminated until there is one with more votes than the sum of the remaining votes. Table 2.8 exemplifies a three round election.

Candidate	Vote
Alice	15%
Bob	35%
Charlie	10%
Eve	40%

Candidate	Vote
Alice	25%
Bob	35%
Eve	40%

Candidate	Vote
Alice (winner)	60%
Eve	40%

Example 2.6: A three round election

Preferential voting Similarly to the *Weighted multiple candidate selection*, in the *Preferential voting* the voter also has the opportunity to give to each candidate a different preference. Ordinarily, the voter order the list according to his preference, as we can see illustrated in table 2.7.

Candidate	Vote
Alice	1
Bob	4
Charlie	3
Eve	2

Example 2.7: *Preferential* ballot example

Single transferable voting This kind of voting mixes the last two voting types and allows multiple winners. It is seen as the most fair voting type, since it minimizes the waste votes, i.e. votes that were discarded since did not choose any of the winners. It has several counting methods, depending on goal of the election. The following example simulates an election with this kind of voting [1].

Imagine an election to nominate the three best chess players of the world. From a set of 10000 voters, each one of them fills a preferential ballot and submit to a ballot box. Then, a quota must be defined. In this case, it will be the Droop quota:

$$votes\ needed\ to\ win\ = \frac{valid\ votes\ cast}{winners + 1} + 1$$

[1] Flash animation: http://archive.fairvote.org/media/bc-stv-full.swf.

Thus, the quota will be 2501. Then, a count is made for each preference and the result will be the first table of Example 2.8. Since only one person has win, the surplus of votes will divided according to the preferences of the extra votes. In this case the surplus was 3000, in which 2001 votes have as second preference Dave and 999 Eve. Dave will be the second winner with 3001 votes, and the 500 extra votes are equally divided between Bob and Charlie. Because no winner is found, the candidate with less votes is removed and his votes are divided between the other two final candidates. In the end, Eve has more votes than Bob, so she is the third and last winner.

Candidate	**Vote**
Alice	5501
Bob	2000
Charlie	699
Dave	1000
Eve	800

Candidate	**Vote**
Alice (win)	2501
Bob	2000
Charlie	699
Dave	3000
Eve	1799

Candidate	**Vote**
Alice (win)	2501
Bob	2250
Charlie	949
Dave (win)	2501
Eve	1799

Candidate	**Vote**
Alice (win)	2501
Bob	2250
Charlie (lost)	0
Dave (win)	2501
Eve	1799

Candidate	**Vote**
Alice (win)	2501
Bob (lost)	2408
Charlie (lost)	0
Dave (win)	2501
Eve (win)	2590

Example 2.8: Single transferable vote

2.2.2 Generic Model

To give a general overview of how a voting system works, we present a generic model based in some different schemes, studies and projects[Mägi, 2007, Ryan et al., 2009, Sampigethaya and Poovendran, 2006, Mercuri, 2001, Cranor and Cytron, 1997, Aditya et al., 2004b]. A general view of the activities involved in a election is summed up in Figure2.1.

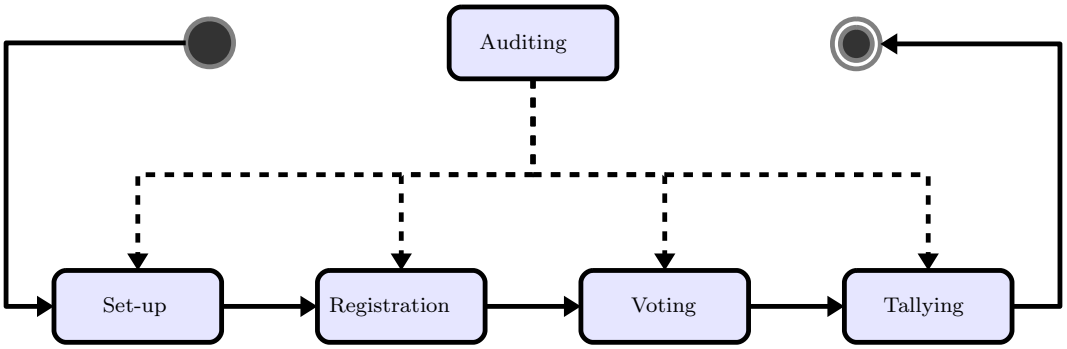


Figure 2.1: General diagram of voting activities

The next list describes each stage. It is important to note that this is an attempt to describe the behavior of an election in the most abstract possible way, in order to cover all voting schemes. Thus, the technical stages that generally fall into *e*-voting systems (e.g. data storage) will not be described.

Set-up is also known as *Announcement* stage, is the initial stage where the voting parameters are initialized. This covers the candidates, the voters, the authorities, the type of the election (e.g. counting rules or ballot validity rules) and voting procedures.

Registration is the phase where is generated the election list, i.e. the public list with all the eligible voters. For this purpose, each voter must register accordingly to different criteria defined in the previous stage, which will determinate their eligibility.

Voting starts with the authentication of the voter. If he is an eligible one, he will have access rights to vote. To do so, first the user needs to authenticate, using some kind of credentials, for example, an identity document. Afterward, there are two steps to a voter be able to cast his vote:

- Voter choice - the step in which the voter picks his candidate. It should be done in a private place, to avoid any kind of intimidation or coercion by a malicious party.
- Vote submission - in an anonymously way, to ensure that his vote is untraceable, the voter save his vote. In the secret paper ballot, this is done by folding the ballot and inserting it in a sealed ballot-box.

Tallying is the set of the actions necessary to compute the voting final result. To achieve the result, it goes through three processes:

- Ballot Collecting - after the voting stage, the ballots are gathered. This process varies depending on the technology that is being used. For instance, in a secret paper ballot, the ballots are collected after an authorized entity opened the sealed ballot-box.
- Ballot Validation - each one of the votes collected previously is validated. This way only correct ballots are counted in the final result.
- Compute/publish tally - the valid votes are counted and tabulated, accordingly to the counting rules (as seen in Subsection 2.2.1). Later on, after the results of each polling station is aggregated, the final result is published.

Auditing is a phase to check the good functioning of the system. It may occur in parallel with the election and it is done by an authorized entity, which has, as an assignment, the following:

- Verifying the Announcement stage - check if the set up has been done correctly. One example may be verifying the existence of bias regarding candidates.
- Verifying the Registration stage - verifies if every possible eligible voter is in fact eligible.
- Verifying the Voting stage - see if every vote has been made according to the rules.
- Verifying the Tallying stage - verifies if the final tally is the right result. For instance, may check for signs of double voting.

2.2.3 Entities Involved

In addition to the previous definitions and descriptions, it is also important to describe a few entities that belong in most voting systems. Some of these entities are not mandatory, some only take part in a voting system if its task is needed, and some may even make the role of others beyond their own. With this in mind, the next list provides a overview of the most common entities.

- Voter - the voter is the main entity that belongs in every voting system. One should not try to predict his behaviour and model a system based on that. Instead, the system should be prepared to any kind of action/behaviour of the voter, being or not ethical.
- Authority - the authorities are the entities responsible to manage, secure and judge elections. A common mistake when modelling a voting system,

which leads to a great lack of security, is to trust in a single entity to be responsible for all of those roles. Therefore, the authorities usually divide into:

  - Administrator - the entity that manage the election. Usually responsible to spread ballot forms, publishing voting related information, recruiting local officials, announcing the final result, etc.
  - Collector - the collector task is to aggregate and tally the votes. A common security requirement is that his relation with the administrator should be collusion-proof.
  - Mixer - this entity is ordinarily a computer server whose only assignment is to shuffle a great number of votes, in order to make them untraceable.
  - Talliers - usually, the tallier and the collector are the same entity. Has the task of processing the votes and compute the final tally.
  - Trustees - the trustees have as a sole purpose to prevent that the security of a system depends on the honesty of only one entity. For instance, in a tallying stage of an election, it could be only possible to count the votes if all trustees are present. Schemes that have only one trustee can be considered susceptible to corruption.
  - Bulletin board - some *e*-voting schemes may need Bulletin Board (BB). This entity is usually a server and its job is to post public information in order to the election be universally verifiable.

- Candidates - the entities that compete in an election. They do not interact directly with a voting system and can assume many different forms, such as political parties, *Yes/No* (for public referendums), real people, etc.

- Auditors - when an election is over, there is a need to guarantee that the system has not been tampered. Hence, a special entity, in this case the auditor, has the job to audit and verify every voting phase, checking for system anomalies and collusions of the other entities. His work is inversely proportional to the level of security of the system. They are also known as *scrutineers*.

- Adversaries - the adversaries are the malicious entities that must be considered in any system. For instance, they could be:

  - Corrupted entity - one of previous entities that interferes or modifies the system only for personal gain;

- Crackers - malicious hackers that attack the system at the behest of someone;
- Coercers - entities that force voters to vote in a certain way;
- Bribers - someone that pays a voter to corrupt his judgement, in this case the ballot.

2.3 Voting Requirements

Throughout the time [Jones, 2009, Grimm et al., 2006, Gerck et al., 2002, Adida, 2006], the requirements of voting systems have been a subject of discussion. As a result, there are considerable differences in the voting systems available on the market, designed and conceived according to the needs that the developers considered more important. Some opt to give more importance to requirements related to accessibility or mobility, e.g. "the voting system must be simple to understand and operate", some choose requirements regarding security above all, e.g. "the voting system must guarantee anonymity of the voter , and some even choose to specialize in different kind of elections. Nevertheless, there is some agreement regarding the basic requirements.

In 1993, Michael Shamos [Shamos, 1993] suggest, in form of commandments, a few fundamental security requirements for *e*-voting systems, listed next in decreasing order of importance:

i. Thou shalt keep each voter's choices an inviolable secret.

ii. Thou shalt allow each eligible voter to vote only once, and only for those offices for which she is authorized to cast a vote.

iii. Thou shalt not permit tampering with thy voting system, nor the exchange of gold for votes.

iv. Thou shalt report all votes accurately.

v. Thy voting system shall remain operable throughout each election.

vi. Thou shalt keep an audit trail to detect sins against commandments *ii*-*iv*, but thy audit trail shall not violate commandment *i*.

Each one of the six commandments refers to one (or more) security properties. Some of those properties are still debatable, whether they are relevant to a voting system, or even if one contradicts another. The forthcoming subsections present a set of the most important properties of an *e*-voting system, the first regards the security ones and the latter is related to the general requirements that are usually evaluated in a *e*-voting system.

2.3.1 Security properties

The following set of definitions were chosen taking into account its informality, abstraction and acceptance. This is because some authors diverge on the definitions, and even adapt them to their own voting schemes.

Correctness *Voting schemes must be error-free. The votes must be correctly recorded and tallied. Votes of invalid voters should not be counted in the tally.* [Sampigethaya and Poovendran, 2006]
This property is also known as *Accuracy*. An underlying property can be deducted from it:

- Integrity: *in the context of an election system is the property that guarantees that the result of the election is not manipulated or altered in any way. This means that all the steps involved in processing the votes preserve the actual information and not a tampered one.* [Ryan et al., 2009]

Privacy *In a secret ballot, a vote must not identify a voter and any traceability between the voter and its vote must be removed. Maximal privacy is achieved by a voting scheme, if the privacy of a voter is breached only with a collusion of all remaining entities (voters and authorities).* [Sampigethaya and Poovendran, 2006]

Uncoercibility In order to avoid coercion, a voting system needs to satisfy two properties, *Receipt-freeness* and *Coercion-resistance*. Although the goal of these properties is the same, both meet different criteria, as defined by [Ryan et al., 2009]:

- Receipt-freeness: *is the requirement that voters are not able to prove to a third party how they voted. In other words, voters should not have, or be able to generate, evidence of how they voted. This is important to avoid vote selling, or demonstrating to a coercer after the election that the voter has voted in a particular way. A receipt can provide evidence that some vote was cast, but not which vote was cast.*
- Coercion-resistance: *means that the system provides mechanisms that would foil any potential coercer, who is in a position to require a voter to vote in a particular way. Even if the voter is interacting with the coercer during the voting process, the coercer should not be able to establish whether the vote was cast in the way demanded.*

Verifiability is the need to verify if the results are correct. It can be expressed in two forms:

- Individual Verifiability (a.k.a. Ballot casting assurance) *Each eligible voter can verify that his vote was really and correctly counted.* [Rjaskova, 2002]
- Universal Verifiability *Any participant or passive observer can check that the election is fair, i.e. the published final tally is really the sum of the votes.* [Rjaskova, 2002]

Later, in [Ryan et al., 2009], the authors had the need to add a notion of End-to-End (E2E) verifiability, which means that the verification, individual or universal, can be made by anyone. This is the strongest notion of verifiability, and currently is the main goal of many *e*-voting schemes.

Robustness *This is concerned with resilience in the face of random faults as well as deliberate attempts to disrupt the election, such as denial of service attacks. One aspect of this is an ability to recover from cheating when it is detected. Another aspect is the ability to run the election even in the face of a minority of dishonest election authorities, for example tellers refusing to decrypt ciphertexts, or mix servers failing to operate. Techniques such as fault tolerance, threshold cryptography and voter-verifiable paper audit trails can be used to provide robustness.* [Ryan et al., 2009]

Democracy is a set of security properties related to the proper functioning of a voting system:

- Prevention of Multiple Voting: *This ensures that all voters are allowed to vote only once, such that each voter has equal power in deciding the outcome of the voting.* [Aditya et al., 2004a]
  This property is sometimes known as *Unreusability* or *Uniqueness*.
- Completeness: *A secret ballot protocol is said to be complete if the ballot of an eligible voter is always accepted by the administrator.* [Juang and Lei, 1997]
- Eligibility: *Only authorized voters are allowed to vote, preventing fraudulent votes from being counted in the tally stage.* [Aditya et al., 2004a]

Fairness *No participant can gain any knowledge, except his vote, about the (partial) tally before the counting stage (The knowledge of the partial tally could affect the intentions of the voters who have not yet voted).* [Rjaskova, 2002]

2.3.2 General Requirements

Adding to the previous security properties, there are also some non-functional and general requirements to improve the quality of a voting system. The next list describes some of the most considered requirements:

Usability *This property is the set of functional requirements related to user-interfaces. For instance, a vote-casting interface should protect the voter from accidentally cast a wrong vote.* [Mercuri, 2001]

Auditability Every system should offer measures to an entity be capable of checking the correctness and integrity of an election. This entity could be a specialized one, but the ideal is for everyone, regardless of being an active part or not of the election, be able to audit the system. This ideal is also known as *open-audit*.

Efficiency *When evaluating a system with this property, one must consider, regarding the technologies and schemes associated to the system, aspects of efficiency, e.g. storage complexity or communication complexity.* [Otsuka and Imai, 2010]

Scalability *The complexity of the protocols used in a voting scheme, is a major factor in its practical implementation. An efficient voting scheme has to be scalable with respect to storage, computation, and communication needs as a fraction of the number of voters.* [Sampigethaya and Poovendran, 2006]

Availability *Since there is a timely nature of elections, availability for the duration of the voting session, as well the tabulation period, is critical. At any time, some kind of backup should be available.* [Mercuri, 2001]

Flexibility (or Maintainability) *The scheme is adaptable in terms of number of authorities or security/efficiency trade-off choice. It should also be able to choose from all kind of tallying methods.* [Sako and Kilian, 1995]

Fault Tolerance *A voting system must ensure that certain capabilities (such as ballot data retention) remain available in the event of failures (like power failure, hardware failure, or software error). The voting system must be capable of remaining in a secure state after a failure so that its relevant security policies continue to be enforced.* [Mercuri, 2001]

Convenience *A system is convenient if it allows voters to cast their votes quickly, in one session, and with minimal equipment or special skills.* [Cranor and Cytron, 1997]

Mobility *A system is mobile if there are no restrictions (other than logistical ones) on the location from which a voter can cast a vote.* [Cranor and Cytron, 1997]

2.4 Summary

This chapter gave an overview of the *e*-voting world. Section 2.1 justifies the need of research in this area, by showing numerous advantages of choosing electronic elections over other technologies, specially paper ballot, and the difficulties of using it. As stated in [Gerck et al., 2002], the *e*-voting technology is still far from being universally acceptable. Other voting technologies, like optical scan (see Appendix A.1), offer reasonable balances of security, ease of use, cost, simplicity and reliability.

A conceptual perspective of *e*-voting elections has been given in Section 2.2. It starts by presenting some of the most used voting types. It is important to note that a good voting system must provide some of those voting types. Then, a generic model of an election is presented and tries to cover all the election stages. For last, some entities that usually belong in voting systems are presented.

The final section talks about voting requirements. The general requirements and the security properties are defined according to the most complete definition found in literature. This consisted in a big problem, since there is no complete standard list for voting requirements. Thus, for each property found, it was decided to choose the definition most concise and cited. A common problem during this stage was finding properties that contradict other properties. In the next chapter, the reader will see how cryptography enters in *e*-voting schemes to achieve the properties discussed in this chapter.

Chapter 3

Cryptographic Primitives

Let us never forget that government is ourselves and not an alien power over us. The ultimate rulers of our democracy are not a President and senators and congressmen and government officials, but the voters of this country.

Franklin D. Roosevelt

Lately, the voting system world is trying to enter in the digital information era. But one of the main reasons that it had not achieved such goal, is the lack of security concerning it. There are numerous universities, companies and investigators trying to create the ultimate system. One of the resources that many of them rely on, is cryptography, which plays an important role on most of the voting schemes proposed so far.

The goal of this chapter is to give the reader a decent background on cryptographic primitives focusing on the three most followed philosophies when creating Electronic Voting (*e*-voting) protocols:

- Based on Homomorphic Encryption (HE), in Section 3.1;
- With the use of Blind Signature (BS), which is described in Section 3.3;
- And finally, using Mixed Network (mixnet), as seen in Section 3.2.

Later, Section 3.4 will explain other primitives that are recurrently used, but without the same frequency. Finally, an example of an *e*-voting scheme will be presented in Section 3.5 for each one of the main primitives. It is recommended that the reader that has good knowledge of the area proceed to the next chapter.

3.1 Homomorphic Encryption

Homomorphic Encryption (HE) is a kind of encryption used when one wishes to compute some specific operations over a ciphertext which returns the same ciphertext as the one returned by the encryption of the result of those same operations over the plaintext. As stated in [Fontaine and Galand, 2007], there are a lot of different situations where this kind of encryption could be proved useful, for instance: *e*-voting, electronic auctions, lottery protocols, watermarking or fingerprinting protocols, secure multi-party computation, etc.

As an example, consider the delegation of computation to an untrusted computer. Thanks to HE, it is possible to achieve the following scenario, where one has two messages, x_1 and x_2, and wants the untrusted computer to compute $x_1 \cdot x_2$, as seen in Figure 3.1.

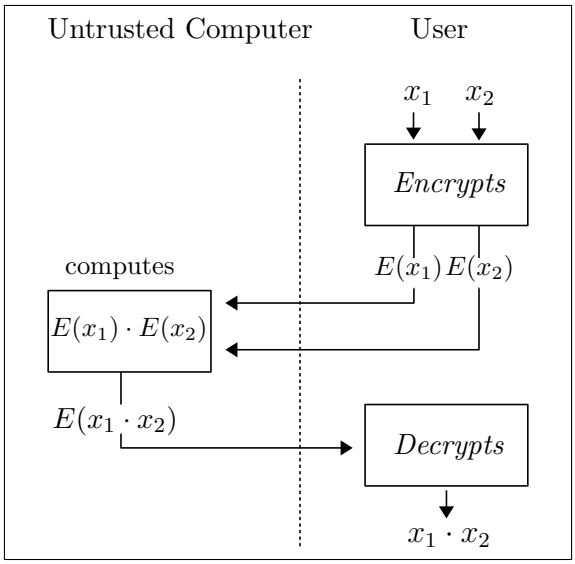


Figure 3.1: The user delegates, safely, mathematical computations to the untrusted computer.

In this case, the untrusted computer will never learn the values of x_1 and x_2, as well as the value of its multiplication. Of course, it is hard to achieve both security and fully homomorphic properties in the same encryption algorithm. But there are some well known cryptosystems that preserve homomorphic properties for some operations, for instance, the Rivest, Shamir and Adleman (RSA) signature described in 3.3. Another cryptosystem that is partially homomorphic is the ElGamal, as can be seen in the next subsection.

3.1.1 ElGamal

The ElGamal Encryption (ElGamal) was created by Taher Elgamal in 1984, at the same time as the less known ElGamal signature, both published in [El Gamal, 1985]. It is an algorithm on the asymmetric key encryption category and its security depends on the hardness of resolving the Discrete Logarithm Problem (DLP) (see Appendix A.2.1).

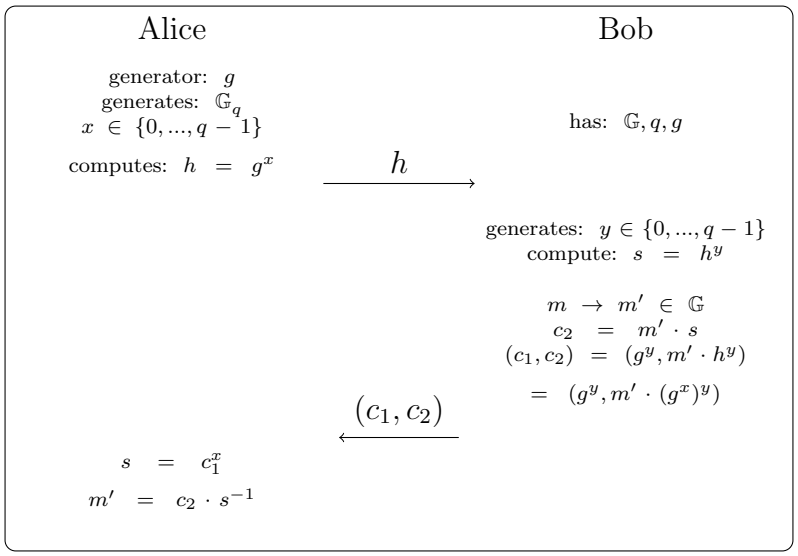


Figure 3.2: ElGamal Encryption

As shown in Figure 3.2, the ElGamal is a scheme between two parties, Alice and Bob. This kind of Public-key Cryptography (PKC) consists of three stages:

- *Key generation* - After the generation of a multiplicative cyclic group $\mathbb{G}$ of order q and with generator g, Alice chooses a random x, which will be the private key, from $\{1, ..., q-1\}$ and computes $h = g^x$. Finally, she publishes h along with $\mathbb{G}, q, g$.
- *Encryption* - the next step starts with Bob choosing a y from $\{1, ..., q-1\}$ and computing $c_1 = g^y$. He proceeds to calculate the shared secret $s = h^y$ and converts the message m into a element m' of $\mathbb{G}$. The last action of Bob is to the second commitment $c_2 = m' \cdot s$ and send the ciphertex (c_1, c_2) to Alice.
- *Decryption* - decrypting the ciphertext involves the calculation, this time by Alice, of the shared secret $s = c_1^x$. Afterwards, she computes $m' = c_2 \cdot s^{-1}$ and converts it into the plaintext message m.

The multiplicative homomorphism property, as the one needed by the *User* in Figure 3.1, can be proven in ElGamal:

$$E(m_1) \cdot E(m_2) = (g^{y1}, m_1 \cdot h^{y1}) \cdot (g^{y2}, m_2 \cdot h^{y2}) = (g^{y1+y2}, (m_1 \cdot m_2)h^{y1+y2}) = E(m_1 \cdot m_2)$$

3.1.2 Exponential ElGamal

The Exponential ElGamal Encryption (EEG) was created due to the need of additive homomorphism in several protocols[Adida, 2006]. The difference from the standard ElGamal lies on minor changes in the Encryption and Decryption stage:

- *Encryption* - instead of m, Bob uses $g^{m'}$ when creating c_2, which results in $g^{m'} \cdot s$. The stage ends in the same way, with Bob sending the ciphertext (c_1, c_2) to Alice.
- *Decryption* - decrypting is also similar with the regular ElGamal. The additional step is to calculate the discrete logarithm of $c_2 \cdot s^{-1}$.

Again, an homomorphism, this time the additive, can be proven using EEG:

$$E(m_1) \cdot E(m_2) = (g^{y1}, g^{m_1} \cdot h^{y1}) \cdot (g^{y2}, g^{m_2} \cdot h^{y2}) = (g^{y1+y2}, (g^{m_1+m_2}h^{y1+y2}) = E(m_1 + m_2)$$

Other additive homomorphism can be seen in the Paillier Encryption scheme (appendix A.2.2).

3.2 Mix-Nets

Introduced by David Chaum in 1981 [Chaum, 1981], mixnet has been a great solution for many security problems regarding anonymity. Roughly speaking, mixnet can be seen as a black box that takes a sequence of messages as an input and returns it encrypted and in a different order. In the world of *e*-voting, it turns useful to disguise the order of incoming of votes. Thus, through a series of permutations from several different black boxes, the order of incoming votes becomes practically irrelevant, decoupling the voter from the vote and providing anonymity.

Figure 3.3 shows the case of a message m and its transformations when passing through a mixnet. Each box, under the name of E_x, corresponds to a *mixer*, as the one described in Subsection 2.2.3, responsible to permute each

received message and encrypt it with his own key. Hence, m goes from the fifth position to the first and becomes $m_1 = E_1(m)$. Then, in the *mixer* E_2, goes to the fourth slot, and it is encrypted again. And so on, until the final *mixer* returns the message $m_3 = E_3(E_2(E_1(m)))$.

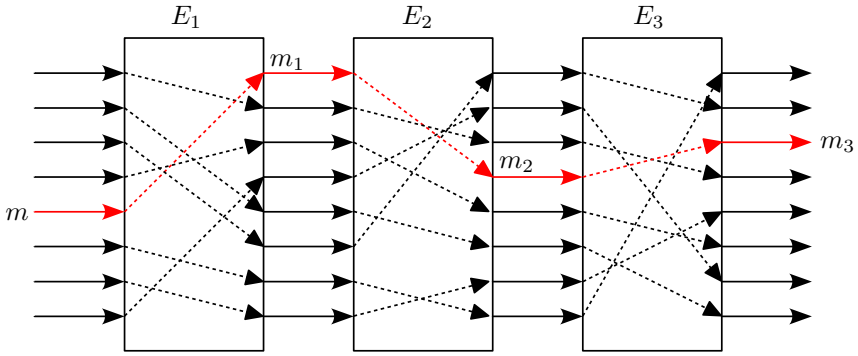


Figure 3.3: Mix-net example.

The biggest advantage of this cryptographic primitive is the joint work of several *mixers* and the level of trust that comes from them. For instance, from K *mixers*, if $\mathsf{K}-1$ are dishonest and collude with each other, the mixnet will still be provably secure if the remaining *mixer* does his job honestly and correctly. This primitive does not exist on its own, i.e. it is the joint work of several other primitives, such as HE, Zero-Knowledge Proof (ZKP) or re-encryption, that create mixnet. In the work of [Adida, 2006], one can find a dedicated space to mixnet, together with a complete table of compared designs of this still well researched primitive.

Additional Notes

Overcoming the anonymity, or untraceability, problem is the main purpose of mixnet. But in some cases, like *e*-voting schemes, it is needed a proof of correctness. This requirement usually adds complexity to the design of mixnet and affects the scheme on which it is involved. To help the evaluation of such schemes, a categorization of the security level of mixnet is made, as stated in [Adida, 2006].

Levels of privacy. Regarding the privacy of mixnets, since its design depends on simpler cryptographic techniques, i.e. standard encryption algorithms or more common primitives, its security will depend on computational assumptions of those techniques. Thus, one has to evaluate the level of the

proofs, which are defined according to the amount of leakage that the proof may have:

- *Complete & independent* - taking Figure 3.3 as an example, message m can be permuted to any other slot. Additionally, if a subset of transformations between input and output of a *mixer* is revealed, the proof of correctness will not affect the remaining messages.
- *Complete but dependent* - differs from the previous level in a way that if the subset is revealed, additional correspondences may be leaked by the proof.
- *Incomplete* - the proof leaks information about correspondences, for instance, when proving the shuffling on E_1, a verifier may know that m has gone from the fifth slot to the first.

Levels of soundness. Proving the correctness of a mixnet can be made using ZKP (will be described in 3.4.3) or other mechanisms that a verifier may learn something about the shuffling. If using ZKP, a prover with bounded computational resources has a negligible chance of successfully proving an incorrect shuffle. Otherwise, a regular prover may have some more probability cheating a verifier.

This issues are both linked and affects the usability of a mixnet. Achieving the maximum level of security is hard and if achieved, it may be proved useless due to high computational requirements. Nevertheless, in *e*-voting, for keeping the election results uncompromised, because of the large amount of messages involved, an intermediate level of privacy can be enough without compromising the election results.

3.3 Blind Signatures

In literature can be found numerous ways to explain blind signatures, generally making use of analogies. The best way to illustrate the concept is with an example taken from the original paper written by by D. Chaum when he first introduced it (can be found in [Chaum, 1982]).

In a voting election by secret ballot, in which the voters send their vote by mail, there is an entity called *trustee* who is responsible for managing the election. His concern is to ensure everyone who votes are authorized to do so. On the other hand, each voter is concerned about the privacy of his vote, even

from the *trustee*, that one malicious entity may be able of corrupting it, and also the possibility that his vote is not counted.

The solution to this problem is to combine two envelops: One regular envelope and one made of carbon paper[1] inside it (analogy of the blind signature's implementation). The protocol between the voter and the *trustee* is split in three stages:

i) In the first stage, the voter fills a ballot with a random *id* and puts it inside of a *carbon paper envelope*, which in turn will be inside of a regular envelope (envelope A). He then writes his address and the *trustee*'s address in envelope A and sends it.

ii) The second stage begins with the *trustee* checking if the sender can vote. If he does, the *trustee* opens envelope A. Next, he discards envelope A and signs the *carbon paper envelope*, which is implying that the ballot inside the *carbon paper envelope* will be marked with his signature. He puts it inside a new envelope (envelope B) with the voter address and sends it.

iii) Finally, the voter opens envelope B and discards it, opens the *carbon paper envelope* and discards it, put the ballot in a anonymous envelope (envelope C) with only *trustee*'s address and sends it to him.

When all voters have been through this process and sent their envelope C, the *trustee* gathers all the votes and verifies each one of them, searching for his signature. He then publishes the results and put in public display all the votes.

At the end, the voter's concerns are resolved because they can see if their vote is right and because *trustee* could not, in any moment, violate the voter's privacy and check for it (the voter search for his vote by its random *id*). In turn, the *trustee*'s concern is also resolved because only the votes with his signature are validated.

Blind RSA Signature

The RSA algorithm is one of the most known and used algorithms for PKC, which was invented in 1977. Its use extends to Digital Signature (DS) and its security is based on presumed difficulty of the factoring problem (see Appendix A.2.1).

[1] For better plainness: `http://en.wikipedia.org/wiki/Carbon_paper`.

As shown in Figure 3.4, a standard RSA signature scheme usually consists on three phases: the *request*, the *signing* and the *verifying* phase. A *Signer* generates a large public number n product of two large and secret primes p and q, and a pair of keys (e, d), the public key and the private key respectively. It is also important to mention that the private key d is the multiplicative inverse of e modulo $(p-1)(q-1)$.

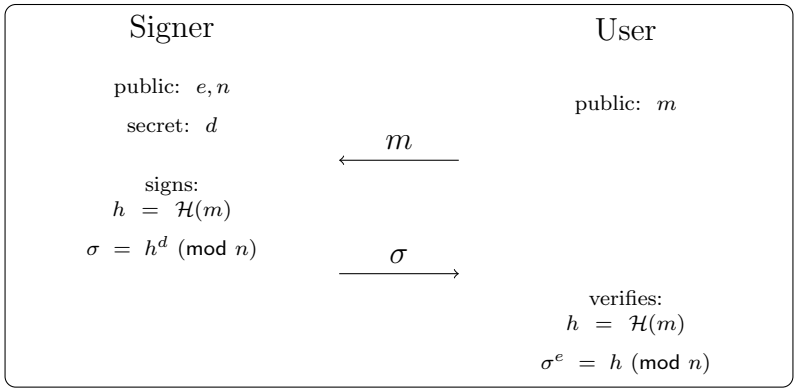


Figure 3.4: Classical RSA Signature

The scheme starts with the *Signer* and a user agreeing on a public *hash function* H. Then, the user makes a *request* by sending a message m to the *Signer*. Next, the *Signer* creates a signature σ by *signing* m, $\sigma = H(m)^d (\mathsf{mod}\ n)$, and sends it to the user. Finally, the user checks its validity by *verifying* the equation $\sigma^e = H(m)\,(\mathsf{mod}\, n)$.

The blind RSA signature differs from the regular DS on the *request* and *verifying* phases. In the *request* phase, instead of sending m to the *Signer*, the user "blinds" it by creating and sending $m' = H(m) \cdot r^e (\mathsf{mod}\, n)$. The *Signer* normally signs m' creating signature $\sigma' = m'^d (\mathsf{mod}\, n)$ and sends it to the user. The user extracts σ by dividing σ'/r. This works by *verifying* that $\sigma = \sigma'/r = m'^d / r\,(\mathsf{mod}\, n) = (H(m) \cdot r^e)^d / r\,(\mathsf{mod}\, n) = H(m)^d \cdot r^{ed} / r\,(\mathsf{mod}\, n) = H(m)^d$. Figure 3.5 shows the transformation of the RSA signature.

Another example of blind signature transformations, specifically of the Schnorr Signature, can be found in appendix A.2.4.

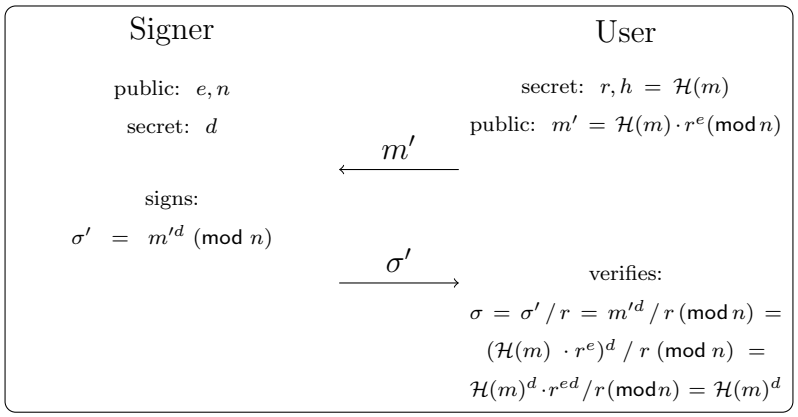


Figure 3.5: Blind RSA Signature

3.4 Other primitives

Generally, the three previous primitives are the most required when designing new *e*-voting schemes. But in most cases, they do not answer to all the requirements needed of those designs. In this section, a brief description of some other primitives will be given.

3.4.1 Re-Encryption

Re-encryption is a technique used to add new secrecy (or re-randomization) a ciphertext without violating the integrity of the plaintext. In *e*-voting, its use is normally related to the use of mixnet. As seen in Figure 3.3, the permutation of the first *mixer* will be useless if $m = m_1$, i.e., matching the input and output of the first *mixer*, one could easily conclude how the permutations occurred.

Thus, to achieve new secrecy, a re-encryption of m is needed. An example of an encryption algorithm that achieve it, described more completely in [Boneh and Golle, 2002], is ElGamal. Remember (see Subsection 3.1.1) that a ciphertext returned by this cryptosystem is $E(m; r) = c = (g^r, m \cdot y^r)$, in which the secret key is x and the public key is $y = g^x \bmod p$. Due to the homomorphic properties of ElGamal, previously demonstrated, the re-encryption of c with a new secrecy r', i.e. $RE(c; r')$, is equal to $c \cdot E(1, r')$, i.e. $E(m; r + r')$. Next, is the demonstration of this fact:

$$\begin{aligned} E(m; r + r') &= RE(c; r') \\ (g^{r+r'}, m \cdot y^{r+r'}) &= c \cdot E(1; r') \\ (g^{r+r'}, m \cdot y^{r+r'}) &= (g^r, m \cdot y^r) \cdot (g^{r'}, 1 \cdot y^{r'}) \end{aligned}$$

$$(g^{r+r'}, m \cdot y^{r+r'}) = (g^r \cdot g^{r'}, m \cdot y^r \cdot 1 \cdot y^{r'})$$
$$(g^{r+r'}, m \cdot y^{r+r'}) = (g^{r+r'}, m \cdot y^{r+r'})$$

3.4.2 Deniable Encryption

The concept of Deniable Encryption (DE) was explored in [Canetti et al., 1996]. Imagine the classical secret communication between Alice and Bob, which exchange encrypted messages with one another. Then, Eve, the eavesdropper, registers the traded messages from Alice and later forces her to give the secret key of the communication. Alice, without any other option, is forced to reveal the key and all the secret conversation between her and Bob is ruined.

The previous situation will probably happen without DE. But with this primitive a new end to this story could happen. If the communication between Alice and Bob was made recurring to DE, when forced to give her key, Alice could provide a fake one. Thus, when Eve use it to decrypt the messages that she stolen from Alice, she would decrypt clear messages but not the real messages traded by Alice. A well known deniable encryption algorithm is the *One-time Pad*, i.e., for any ciphertext produced by this cipher, one can create several valid messages and keys that result on the same ciphertext, as long as it has the same length.

There are several different applications to DE, for instance multiparty computation. To *e*-voting, its use is dedicated to prevent coercion (see Subsection 2.3.1). DE allows the voter to create a fake receipt to fool any adversary that tries to coerce him.

3.4.3 Zero-Knowledge Proofs

A recurrent problem in security systems is proving knowledge of something without revelling crucial information. To resolve such problems, a cryptographic primitive was invented under the name of Zero-Knowledge Proof (ZKP). It is a protocol between a *prover* and a *verifier*, and it states that a ZKP is a proof that yield nothing beyond the validity of the assertion. Thus, a *verifier* only gains conviction in the validity of the assertion[Goldreich, 2002]. A specific type of ZKP are the interactive zero-knowledge proof-of-knowledge, which consists in a sigma protocol (three communications) between two parties: the *commitment*; the *challenge*; and finally the *proof*. In appendix A.2.3, a story helps to understand the ideas of ZKP.

There is not a single ZKP protocol. Each one is constructed according

to specific properties of what is intend to prove. But all correctly formed protocols obey to the following three properties:

- *Completeness* - if the statement is true, a honest *verifier* will be convinced of that fact by an honest *prover*.
- *Soundness* - if the statement is false, no dishonest *prover* can only convince the honest *verifier* that it is true with very small probability.
- *Zero-Knowledge* - most importantly, in any case, no dishonest *verifier* can learn anything about what is intended to prove.

Chaum-Pederson protocol. Figure 3.6 shows an example of a ZKP protocol[2] introduced in [Chaum and Pedersen, 1993] and it is used to prove the existance of a Decision Diffie-Hellman (DDH) tuple, or more precisely, the equality of discrete logarithms.

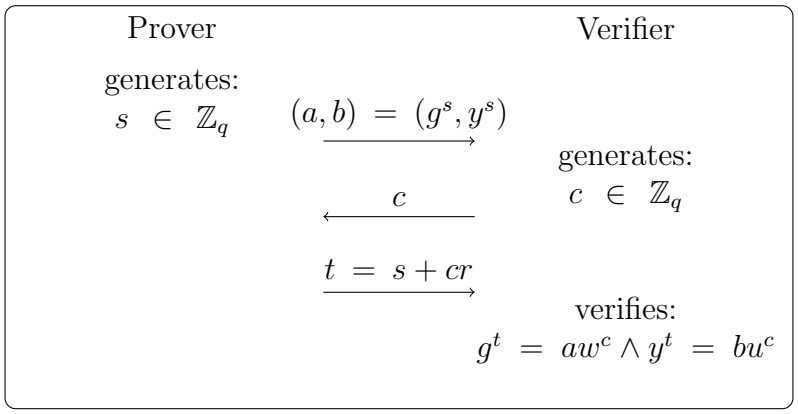


Figure 3.6: Proving the equality of two discrete logarithms.

This protocol has utility if one, for example, wishes to prove the re-encryption of a message m. Using ElGamal, as in the example of Subsection 3.1.1, one creates the ciphertexts $ct_1 = (c_1, c_2) = (g^t, m \cdot y^t)$ and $ct_2 = (d_1, d_2) = (g^u, m' \cdot y^u)$. If ct_2 is the re-encryption of ct_1, then they are both encryptions of the same message. This means that proving that ct_2 is the re-encryption of ct_1 is the same that proving the tuple $(g, y, \frac{d_1}{c_1}, \frac{d_2}{c_2}) = (g, y, g^{u-t}, \frac{m'}{m} y^{u-t})$ is a DDH tuple of the form (g, g^x, g^r, g^{xr}). A honest *prover* will eventually prove the knowledge of r, which is the randomness used to create ct_2.

[2] An interesting set of ZKP can be found in: http://www.cs.ut.ee/~lipmaa/crypto/link/zeroknowledge/pok.php .

Finally, an example of another ZKP, the Guillou-Quisquater identification scheme, can be found in Appendix A.2.5.

3.4.4 Secret Sharing

When evaluating the safety and security of a cryptosystem, it is normal that all the structure rests on a crucial point. This point is usually a secret key, something unique that is only known by one person. But some systems may require that the secret cannot be trusted to a single person. That is why the idea of Secrete Sharing (SS) was introduced in the cryptographic world.

To see the importance of such primitive, a good example was given in [Schneier, 1996]. Imagine that Alice invents an application to launch nuclear missiles. That application rests on a single key, but Alice does not want a single employee to have it, fearing that he may be crazy and initiate the launch just for fun. Recurring to SS, Alice chooses five employees and gives each one a key. She then creates a mechanism that requires at least three from those five employees to insert their keys and launch the missile. Now, three employees must go crazy, together, to launch a nuclear bomb just for fun.

And Alice can even adopt more complex mechanisms. To an officer, named Bob, that "worths" two employees, is given another key. Since he values more than the regular employees, Alice can adapt the mechanism to launch the missile with two inserted keys if one of them belongs to Bob.

Threshold Cryptosystem. A kind of SS that can do all of Alice's ideas, and more, mathematically is called a threshold cryptosystem. The simple case of wanting a secret code divided into k keys, such that t of them can be used to reconstruct it, is called a *(t,k)-threshold scheme*.

In [Gennaro et al., 1999], they suggest a protocol for distributed key generation, the main difficulty in SS. This protocol is for cryptosystems based on the DLP, such as ElGamal (see Subsection 3.1.1). For instance, it may be used in the *e*-voting scheme that will be presented in Subsection 3.5.1. The protocol, between n *Players*, goes as follows:

Generating x:

1. Each player P_i performs a so-called Perdersen-VFF[3] protocol of a random value z_i as a dealer:
   (a) P_i chooses two random polynomials $f_i(z)$, $f_i^{'}(z)$ over $\mathbb{Z}$, of degree t:
   $f_i(z) = a_{i0} + a_{i1} + ... + a_{it}z^t \qquad f_i^{'}(z) = b_{i0} + b_{i1} + ... + b_{it}z^t$
   Let $z_i = a_{i0} = f_i(0)$. P_i broadcasts $C_{ik} = g^{a_{ik}}h^{b_{ik}} \bmod p$ for $k = 0, ..., t$. P_i computes the shares $s_{ij} = f_i(j)$ and $s_{ij}^{'} = f_i^{'}(j)$ for $j = 0, ..., n$ and sends $s_{ij}, s_{ij}^{'}$ to player P_j.
   (b) Each player P_j verifies the shares received from the other players. For each $i = 0, ..., n$, P_j checks if:
   $$g^{s_{ij}}h^{s_{ij}^{'}} = \Pi_{k=0}^{t}(C_{ik})^{j^k} \bmod p$$
   If the check of i fails, P_j broadcasts a complaint against P_i.
   (c) Each player P_i who, as a dealer, received a complaint from player P_j, broadcasts the values $s_{ij}, s_{ij}^{'}$ that satisfies the previous equation.
   (d) Each player marks as disqualified any player that either:
   - received more than t complaints in step (b) or
   - answered to a complaint in step (c) with values that falsifies the equation from (b).
2. Next, the second step starts with every player building the set of non-disqualified players, which will be known as $\mathcal{U}$.
3. The distributed secret value x is not explicitly computed by any player, but it equals $x = \Sigma_{i\in\mathcal{U}}\ z_i \bmod q$. Then, each player sets his share of the secret as $x_i = \Sigma_{i\in\mathcal{U}}\ s_{ji} \bmod q$ and the value $x_i^{'} = \Sigma_{i\in\mathcal{U}}\ s_{ji}^{'} \bmod q$.

Extracting $y = g^x \bmod p$:

1. Extracting the public shared key is for every qualified player, i.e., every $P_i \in \mathcal{U}$. To do so, each one exposes $y_i = g^{z_i} \bmod p$ via Pedersen-VFF:
   (a) Each player P_i broadcasts $A_{ik} = g^{a_{ik}} \bmod p$ for $k = 0, ..., t$.
   (b) Each P_j verifies the values broadcast by the other players in $\mathcal{U}$, in particular, for each $i \in \mathcal{U}$, P_j sees if:
   $$g^{s_{ij}} = \Pi_{k=0}^{t}(A_{ik})^{j^k}$$
   If the check fails for any P_i, P_j complains against him by broadcasting the values s_{ij},$s_{ij}^{'}$ that satisfies the equation of the generation's step (b) but do not satisfy this previous equation.

[3]This name was picked by the authors to mention the protocol described in [Pedersen, 1992]

(c) Finally the reconstruction of y, with every valid player computing z_i, $f_i(z)$, A_{ik} for $k = 0, ..., t$ in the clear. The protocol ends with every player setting $y_i = A_{i0} = g^{z_i} \bmod p$, and computing $y = \Pi_{i \in \mathcal{U}} y_i \bmod p$.

3.5 Schemes and designs

In literature there are a large variety of proposed *e*-voting schemes. As mentioned before, they mainly follow three philosophies, so in this section we will present one well known scheme for each of them. The only criteria was to choose a scheme that was simple to understand.

3.5.1 Cramer, Gennaro and Schoenmakers scheme

The following scheme is an example of how HE can be used in *e*-voting schemes. This example also uses two primitives, ZKP and SS. There are many other schemes that use HE, such as [Benaloh and Tuinstra, 1994, Benaloh and Yung, 1986].

In [Cramer et al., 1997], it is proposed a scheme that consists in three stages: Initialization; Vote Casting; and Tallying. The protocol involves three different entities, the voters, a set of authorities and a Bulletin Board (BB). A special characteristic of this scheme is that no direct communication between the voter and the authorities is made. Before explaining it, is worth noting that the next description of this scheme only allows a "yes/no" kind of ballot (see Subsection 2.2.1), the authors later explained how to add multiple choices.

Initialization. Before the election starts, all authorities agree on a key generation protocol for SS. Using an ElGamal cryptosystem, each authority receives his part of the shared secret $A_j : s_j$ of s and they all publish the public key $y = g^s$.

Vote Casting. Every voter will now interact with the BB. First, they choose which vote they want to cast, whether it will fall on $m_0 = G$ or $m_1 = 1/G$. Then, they randomly generate $\alpha \in \mathbb{Z}_q$ and create the ciphertext $(x, y) = (g^\alpha, h^\alpha \cdot G^b)$, in which $b \in \{-1, 1\}$. They proceed to cast their vote, sending the ciphertext (x, y) to the BB, along with a proof that the ciphertext is well formed. This proof is the non-interactive version of ZKP known as the Chaum-Pedersen protocol (also described in 3.4.3), and has the objective to compare two discrete logarithms. Hence, they use it to prove $\log_g(x) = \log_h(g/m)$.

Tallying. After the election ends, the authorities, together, check every proof and its validity. They then form the product $(X, Y) = (\Pi_{i=1}^{l} x_i, \Pi_{i=1}^{l} y_i)$. Next, each one of the authorities computes and broadcasts to BB $W_j = X^{s_j}$ together with the correspondent proof $\mathsf{log}_g(h_j) = \mathsf{log}_X(W_j)$. Finally, all the authorities, together, decrypt $W = Y/X^s$. The result of W will be equal to G^T, where T is the difference between the number of "yes" and "no" votes. Thus, the discrete logarithm of W will give the value of T, $T = \mathsf{log}_G(W)$.

HE was a crucial piece on this design. The privacy property only is guaranteed thanks to the homomorphic properties presented in the used techniques, and not a single vote is decrypted alone. Nevertheless, this scheme is not perfect. Some problems regarding coercion (see Subsection 2.3.1) have been pointed out, and also some computations can turn out to be over expensive, in particular the calculation of T.

3.5.2 The Fujioka et al. Voting Scheme

The classical [Fujioka et al., 1993] was one of the firsts to ever use BS in an *e*-voting scheme. Among other schemes, we can find for example [Kim et al., 2001, Ohkubo et al., 1999]. This scheme consists in six stages and has three entities involved: the Voter V_i; the Administrator; and Collector. The next list describes the stages that occur accordingly with Figure 3.7:

1. Preparation - The first stage starts with a Voter V_i creating a ballot x_i, which is the encryption of the vote v_i using a key k_i. He then *blinds* the ballot with a random r_i and creates a message $e_i = X(x_i, r_i)$. Finally, he signs it, $s_i = \sigma_i(e_i)$ and sends (id_i, e_i, s_i) to the election's Administrator.

2. Administration - The Administrator is responsible to validate each request to vote. To do so, he checks if the received id_i is an eligible voter and if he has never requested to vote before. If the Administrator do not find any problems, he signs over the blinded message $d_i\sigma_A(e_i)$ and sends it as a certificate to V_i. Otherwise, he rejects the request. After all the requests, the Administrator publishes a list with every blinded message, identifications and signatures $((id_i, e_i, s_i))$, and starts the voting stage.

3. Voting - Being confirmed as an eligible voter, V_i proceeds to cast his vote. He first retrieves the Administrator signature $y_i = \lambda(d_i, r_i)$. Then, after checking the validity of y_i, he cast his vote by sending (x_i, y_i) to the Collector through an anonymous channel.

4. Collecting - The Collector, after verifying the Administrator's signature from every received pair (x_i, y_i), publishes a list with the correct votes and a number l, i.e., $(l_i, (x_i, y_i))$.

5. Opening - After the collecting stage is over, V_i checks if his ballot is on the list. Then, if it is there, he sends his key k_i along with the respective number l to the collector, also through an anonymous channel.

6. Counting - Finally, using the received k_i, the Collector opens every ballot and publish the results, with every public information $(l_i, x_i, y_i, k_i, v_i)$, and everyone can verify it.

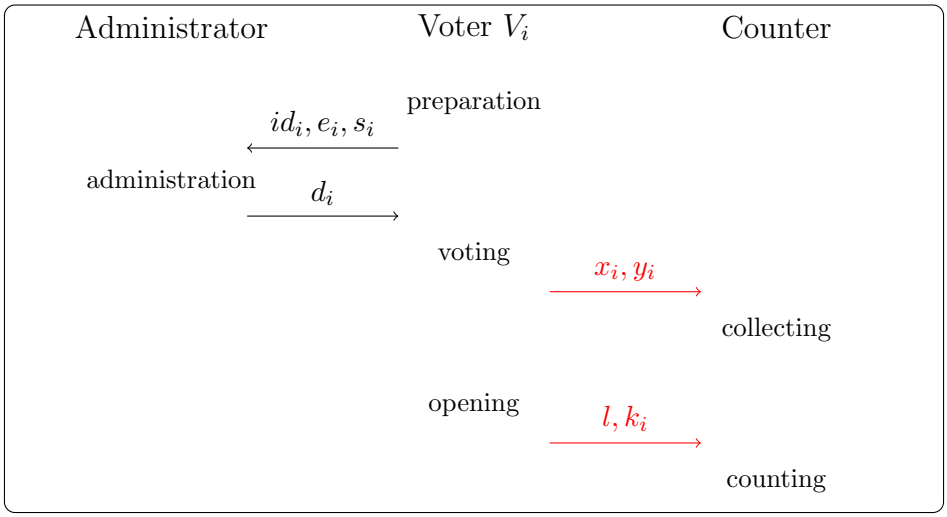


Figure 3.7: Fujioka et al. voting scheme.

This scheme demonstrates the use of BS and has been subject of study and research, as well as inspiration to many other *e*-voting schemes. But over the years some attacks have been discovered against this scheme. For instance, in [Rjaskova, 2002], the author explains how a corrupt Administrator can simulate votes from abstained voters. The same author also shows how the receipt-freeness (see Subsection 2.3.1) is another property violated. And even more problems can be found in implementing the anonymous channels, the last two communications on Figure 3.7. Nevertheless, this scheme is still good to demonstrate the utility of BS.

3.5.3 Boneh and Golle's mix-net

There are plenty of schemes using mixnets, such as [Aditya et al., 2004a, Chaum et al., 2005]. The following scheme, proposed in [Boneh and Golle,

2002], uses several primitives and techniques presented before, namely:

- Re-encryption with ElGamal from Subsection 3.4.1;
- ZKP and the Chaum-Pedersen protocol shown in Subsection 3.4.3;
- SS and a simpler version of the example presented in Subsection 3.4.4.

The protocol consists in five steps, since the setup stage until the decryption of each message submitted. It works between a set of users, *mixers*, and a BB:

Setup. Before protocol begin, a set of *mixers* $\mathcal{M}$ fulfill a key generation protocol for SS of an ElGamal cryptosystem, in order to each M_j has his share x_j of the secret.

Submission of inputs. After the setup:

- The *mixers* publish the public ElGamal parameters (g, q, y);
- Each user submit his ciphertext of the message m encrypted with ElGamal. It other words, each U_i sends his $C_i = (g^{r_i}, m_i \cdot y^{r_i})$, along with a proof of knowledge of m_i, to BB;
- Finally, the *mixers* agree on a security parameter $\alpha > 0$, usually less than five. Higher values, as one will conclude later on step *Verification*, will result in less privacy for the user.

Re-encryption and Mixing. In this stage, each *mixer* will re-encrypt and shuffle the list of ciphertexts, one at a time:

1. M_j takes the set of n ElGamal ciphertexts, $\mathcal{C}$, in the form $C_i = (g^{r_i}, m_i \cdot y^{r_i})$, from the BB;
2. Then, adds new secrecy to all ciphertexts of $\mathcal{C}$, in which C_i becomes $C_i' = (g^{r_i'}, m_i \cdot y^{r_i'})$;
3. Last, M_j returns, to BB, $\varphi_j(\mathcal{C})$, where φ is a random permutation chosen by M_j. The *mixer* M_j is required to remember, and keep secret, the permutation used and the randomness added to the ciphertexts until the verification step is over. Each final set produced will be known as $\mathcal{C}_j$ on the verification stage.

Verification. If any of the *mixers* abort, the protocol is over and no results are produced. If not, they all proceed to jointly generate a random string s which will be used to generate random challenges. The string is generated as follows: Each M_j selects a random string s_j and commits

it. After all commitments are received, the final random string s will be $s = Xor(s_j)$. Afterwards, each one of the *mixers* will be audited by the rest of them:

1. First, each C_i from $\mathcal{C}_j$ is checked to see if it is properly formatted.
2. Then, M_j, using Chaum-Pedersen, proves that $\Pi_{i=1}^{n} m_i = \Pi_{i=1}^{n} m_i^{'}$, i.e. despite of the added secrecy, both ciphertexts were encrypted using the same message.
3. The rest of the *mixers* collaborate to generate α sets $S_1, ..., S_\alpha$, where each set S_i is a subset of indexes $\{1, ..., n\}$. Each set is generated independently from one another, in the following manner: every index $1 \leq k \leq n$ is included in S_i independently at random with probability $1/2$. The randomness is derived from the string s.
4. The sets $S_1, ..., S_\alpha$ are given to mix server M_j.
5. M_j must produce α subsets $S_1^{'}, ..., S_\alpha^{'}$ of $\{1, ..., n\}$ such that, for all $1 \leq i \leq \alpha$, $|S_i| = |S_i^{'}|$ and, using Chaum-Pedersen, $\Pi_{k \in S_i} m_k = \Pi_{k \in S_i^{'}} m_k^{'}$.
6. If M_j fails in the previous step, he is accused of cheating. The remain *mixers* will inspect the transcript of the verification, and if M_j is confirmed as a cheater, he is banned and all the other restart the mixing.

Decryption. Finally, a sufficient number of *mixers* jointly performs a threshold decryption of the final set of ciphertexts and provides the correspondent ZKP of correctness.

3.6 Summary

In this chapter, an overview about cryptographic primitives was given. The described primitives enter in the majority of schemes that have been proposed so far[4].

An interesting observation on the studied schemes, was that mixnet are becoming the favorite choice among the researchers to propose new schemes. BS has lost some importance, probably due to problems regarding anonymity, which ironically can be overcome using mixnet, and thus lost its utility. But

[4]Although an incomplete work and never published, the following article has been of enormous help to understand how the cryptographic world was applied to solve the *e*-voting problem: "Survey on Electronic Voting Schemes" by Fouard, Duclos and Lafourcade.

BS is still a well researched area because of its use in electronic-cash. Moreover, HE is the preferred method when putting theory in practice, as can be seen in the next chapter, in despite of the lack of flexibility and adaptability.

There is no primitive that can solve all security problems, neither a scheme that can be adapted to all kind of elections. In the next chapter, some systems that use some of the above mentioned primitives will be described, and in particular a system that uses HE combined with ZKP and SS.

Chapter 4

Helios

Those who vote decide nothing. Those who count the vote decide everything.

Joseph Stalin

Putting the ideas of the previous chapters in practice is a difficult task. In fact, in the literature, it is hard to find practical implementations of suggested Electronic Voting (*e*-voting) schemes. Moreover, most of those implementations are usually *proof of concept*, and not real systems with real tests in real situations. Even more complicated is to find real systems devoted to remote voting. Some emerging contests are appearing to change this scenario. They propose the creation of remote *e*-voting systems with mandatory requirements that aim to protect the voter.

This chapter starts with a description of a contest by the International Association for Cryptologic Research (IACR), a non-profit organization dedicated to cryptologic research. The rest of the chapter describes the Helios system, the winner of the contest. An overview of this system is given in Section 4.2 and most of the used techniques are described in Section 4.3. Finally, some security assumptions and recent research works on this system are presented in 4.4.

4.1 The IACR Contest

In the Spring of 2008, the IACR board published a call for presentations and demos of *e*-voting systems. Their hope was to replace the mail-based sys-

tem with a capable cryptographic *e*-voting system[1] and thus achieve the goals of having a secure system for its internal elections, to enhance the research in this area and to study the impact on public policy issues to *e*-voting in general.

For a fair competition, the committee came up with a list of the minimum satisfying requirements and evaluation criteria[2]. The basics ones mostly concern usability and met the ones described in Subsection 2.3.2, such as: the system must be user-friendly, at least from the voters' point of view; the system must be easy to maintain and manage; the system must be accessible, available, and reliable. With these requirements, they established the ground rules to accept only usable systems and discard the ones that, even if they were completely safe, were not suitable for this contest (for instance, scalability problems).

More importantly, the IACR and the rest of the committee also listed some security requirements, which match the definitions on 2.3.1:

- Eligible voters must be able to vote exactly once. The system must be able to verify that voters are members of the IACR (match the definition of *Democracy*).
- Individual votes must remain secret. The system must prevent attackers from learning how individual members voted (match the definition of *Privacy*).
- The system must have universal verifiability of the integrity of the election outcomes. Namely, the system must allow open audit, where everyone can verify that only valid votes were counted and the tally is accurate, and every voter can verify that his vote was counted (match the definition of *Verifiability* along with *Correctness*).

Other considerations concerning security, but in a viewpoint of attacks, were mentioned (due to the complexity and difficulty of creating systems that prevents the following attacks, these requirements/properties were not mandatory):

- Denial-of-Service (DoS) attack: it is a kind of attack that attempts to make a system's (machine or network) resources unavailable for the clients. The proposed system should offer measures of resistance to DoS attacks, and making it possible to fix system outages in brief period of time (for instance, re-locating servers, distributed servers, etc).

[1]The call: http://www.iacr.org/elections/eVoting/cfp.html

[2]The complete requirements: http://www.iacr.org/elections/eVoting/requirements.html

- Vote modification: the system should have ways to prevent virus or any kind of malware that has as its purpose modify the client's vote.

- Correlation-based attack: a type of attack that makes correlations between votes. For example, if one knows how someone votes for director, may deduct how he will vote for president. The IACR committee suggests a simple way to avoid this by splitting races.

- Receipt-freeness: as described in 2.3.1, there are problems associated with knowing how one votes. Therefore, the committee values the systems that presents mechanisms that fight these problems. A way suggested, was to allow voters to cast "fake votes" that would be discarded in the tallying stage (see 2.2.2) and therefore not affect the final result. This methology imply that even if the attackers are able to monitor the client's machine in order to obtain the receipt, they would still not be able to prove how that person voted.

- Level of trust: not being exactly an attack, the level of trust is an interesting point of analysis of every *e*-voting system. When modelling such system, it is usually required that at least one entity must be trusted in order to achieve security. Thereby, to not depend on the integrity of a single entity's task, it is recommended by the committee to divide the responsibility into several entities. For instance, vote decryption could be a task shared by several *Trustees*.

Finally, the IACR committee stressed out some differences between how they face an election and how usually the public-sector elections occur. Regarding the *resistance to coercion and vote-selling* properties (see Subsection 2.3.1), even though they are crucial aspects for the public-sector elections, to preserve fairness and democracy, for small companies and private elections a weaker property is required. Nevertheless, a special attention to this problem is needed, as upheld in the previous *receipt-freeness* bullet. Another difference when comparing with public-sector elections, is the absence of *paper trail*, which is normally used to be examined if the need appears. For the committee, an electronic version is enough. Concerning the robustness of the system, it was not demanded a completely full proof solution (for instance, the DoS attack mentioned above), but instead a flexible system that may be able to extend a voting deadline for some time if these kind of attacks happen.

Summing up, the IACR committee were not looking for a solution that could be used in a country elections (in fact, it is rare to find a country that trusts only on an *e*-voting system), but a practical and implemented solution for local use.

4.1.1 The Contestants

In August of 2008, the teams that proposed *e*-voting systems had the chance to present them to a jury. Throughout this subsection, it will be given a brief description of each one of the systems, based on the presentations, technical papers and, when possible, the system itself.

Civitas

From the Department of Computer Science of the Cornell University came the so called Civitas[3]. This system was implemented in *Jif* (a security-typed programming language that extends Java) and it was based on the well cited article [Juels et al., 2005], improving the proposed scheme which led to new technical advances in secure registration protocols and a scalable vote storage systems [Myers et al., 2008].

The design model consists in an ordinary voting model, with the normal stages of an election, and with five kind of agents: the *voters*; the *supervisor*, which administers an election; the *registrar* authorities, that authorizes voters; *registration tellers*, which generates credentials that voters use to cast their votes; *tabulation tellers*, responsible to tally the votes and provide anonymity. Each one of the agents (with the exception of the voters) use an underlying *log* service that records all the information needed to audit the election. Concerning the threat model, they assume that the adversary can always perform any polynomial time computation and may corrupt all but one of each type of election authorities. At some time during the election, the adversary may even coerce on any way the voter and control the network.

Regarding the security requirements, the Civitas team ensure that they satisfy the requested properties, and put the emphasis on how they solve the problem related to coercion. To fight it, they use techniques described in [Juels et al., 2005], which consist in using fake credentials. In some situations, described in [Myers et al., 2008], the voter can undetectably use a fake credential to cast a vote, thus deceiving the coercer. With regard to the anonymity property, they used Mixed Network (mixnet) (section 3.2), operated by the *tabulation tellers*. To resolve the verifiability requirement, they used Zero-Knowledge Proof (ZKP) (Subsection 3.4.3).

Civitas have a list of assumptions to prove the security of the system,

[3]Official site of the voting system Civitas: `http://www.cs.cornell.edu/Projects/civitas/`

specially to maintain the coercion-resistance property. For instance, they assume that the channels from the voter to the ballot boxes are anonymous. Besides these assumptions, some issues were detected and were targeted for improvements. Technical issues include: web interfaces; testing; threshold cryptography; and anonymous channel integration. On the side of research problems: distribute trust in voter client; eliminate in-person registration; credential management; and application-level DoS.

Adder

Adder voting system was developed in the University of Connecticut, and adopts a strong voting-oriented cryptographic primitive, the homomorphic encryption (section 3.1). The system has three components: a bulletin board server, developed in C++; an authentication server, with the back-end also written in C++; and a client software (either a Java applet in a Web-browser or a stand-alone program) [Kiayias et al., 2006].

Three entities interact with the system: as usual, the voters; *authorities*, entities in charge for both maintaining the security and privacy of the election; and *administrators*, which are responsible to create and manage elections. The system hopes to achieve some security properties described in section 2.3.1, namely: universal verifiability; privacy; and *transparency* (not mentioned before, but is described as some kind of End-to-End (E2E) verifiability). The authors also acknowledge that a crucial requirement for remote *e*-voting, coercion-resistance, is not treated, making it an open problem for future upgrades. Another deficiency of this system is the lack of mechanisms for individual vote verification.

Unfortunately, the source-code of this system was not available. Thereby, it was not possible to test the usability and flexibility of the Adder voting system.

Scratch, Click and Vote

This third system was developed in Poland, more concretely, in the Wroclaw University of Technology [Kutylowski and Zagorski, 2008]. The approach of this team was to develop an hybrid *e*-voting system based on previous work by [Ryan et al., 2009](Prêt à Voter), [Chaum et al., 2008b] (Punchscan) and [Cichoń et al., 2008] (ThreeBallot). By being a hybrid voting system, several actions may require to be in-person actions (e.g. it is suggested that the usual registration stage would be in-person), while other actions are made through

electronic means (vote casting). As a result, this may complicate the deployment of the system.

Together with the voter, there are two main entities in this system: the *election authority*, which prepares the ballots; and the *proxy*, responsible for producing the coding cards. Both pieces of paper combined form a complete ballot and using the techniques described in the articles mention above, turns the ballot securely random. When casting the vote on a computer, a virus or any kind of malware will not learn anything from the voter choices. The only cryptographic primitive needed is the signature of the ballot by the *election authority*. To achieve this, they use Blind Signature (BS) (section 3.3).

Besides the difficulties of deploying such system, there are some issues regarding usability. According to the authors, one reason is that the voter must click next to every candidate, to choose whether is he voting for him or not. The other reason is that he needs to find his candidate on a permuted list of candidates. Despite this, most of the security properties are guaranteed. The major drawback is vote-selling, which is possible if the buyer has access to the ballot, the coding card used and the record of a voting session from the voter's computer. The assumption that guarantees the good functioning of the system is that both authorities do not collude. It is also important to mention that this system does not have distributed trust, a consideration made by the IACR committee.

Easiest Election

Completely based on [Ryan et al., 2009], the Easiest Election was developed by a group of *e*-voting enthusiasts, including some authors from the previous mentioned article that worked as advisors. This has the advantage that most voting types described in Subsection 2.2.1 can be deployed. Only the ones with an interaction like *write-in* votes cannot provide privacy to the voters. The system was not complete at the time of the contest, and apparently neither afterwards[4] , but some description of how things would work can be found on the official website[5] and on the presentation.

Concerning the entities involved with this system, one can infer by the fact that is based on the Prêt à Voter system, that this system has: *voters*, the

[4]After trying to test the application without success, and checking that the existing elections had zero casted votes, in `http://vote.easiestelection.com/`, it was concluded that the work was not finished or that it had been moved to another project.

[5]Official website: `http://www.easiestelection.com/`

entity that cast the vote and may verify its correctness later; *election authority*, which is in charge of distributing ballot forms, recruiting local officials, publishing information, and so forth (in an electronic way, they are usually represented as servers); *auditors*, which provide expert opinions on evidence of proper functioning of the system; and *help organizations*, independent organizations that help the voters on non-private matters regarding voting or checking procedures. Cryptographic techniques are used in two occasions. The first is on the creation of ballots, resulting in encrypted ballots with a unique *id*. The second, more complex, happens when a voter submits his vote. Using mixnet (section 3.2), the system shuffles the votes, breaking the link between the voter and the ballot, which results in anonymity for the voter. If it uses a k number of mixnets, the system will be tamper-proof if at least one of the k is honest.

As mentioned before, and differently from other systems, in Easiest Election each ballot is unique. Therefore, an assumption that ballots cannot be forged must be made. The authors refer some other assumptions that were made to prove the correctness of the system, such as privacy of the polling booth in which the voter marks and cast his vote. Another assumption regards the bulletin board, which they assume that the published information is reliable. Finally, an assumption concerning the electoral committee is made. The list of eligible voters provided by them must be correct.

SVIS

Not much of information could be found about the SVIS voting system. It was developed by several members of Japanese companies and, according to the presentation, use mixnets (section 3.2) for handling the anonymity problem. Also accordingly to the presentation, the system, as an *e*-voting solution, is not portable. Among the problems that the IACR reported is the authentication of the voters, which may suggest that an automatic registration was not implemented or that this system would be hybrid.

Punchscan

Punchscan was an international open-source project of a voting system leaded by David Chaum and had the participation of several North American universities. Later, it evolved to Scantegrity[6], which is still a widely used vot-

[6]Punchscan: http://www.punchscan.org/
and Scantegrity: http://scantegrity.org/

ing system. Like some systems previously described, this is an hybrid system. The solution is oriented for optical-scan machines (see section A.1) and does not make use of complex cryptographic primitives, but instead, some tricks to substitute its job[Popoveniuc and Hosp, 2010].

Regardless of being hybrid, hence requiring some in-person operations, this system was presented to inspire remote and electronic versions of itself. Beginning with the entities involved, there are three that interact with the system: as usual, the voter, which authenticates himself as legitimate voter and proceeds to cast his vote according to his preferences; the *election authority*, the entity that manages the election (generation and decryption of ballots); the *candidates*, the competitors that also have the role of auditors, challenging the authority during the course of the election to check for its integrity and consistency.

Roughly speaking, the trick behind Punchscan is on the creation of ballots. Each ballot is composed of two pieces of paper, both marked with the same *id* . On one piece, it is written the list of candidates, with a random tag, in a random order. The tag cannot be seen unless the voter marks it as his choice. Afterwards, a tag is revealed and he may optionally write it on the second piece of paper. To cast his vote, the voter separates both pieces and inserts the one with the marked choice on the optical-scan machine, saving the other piece as a receipt[7]. Note that the receipt does not reveal the choice of the voter, thus preventing vote selling and coercion.

The security properties defined in section 2.3.1 are preserved as long as several assumptions are made. In [Chaum et al., 2008a], the authors divide the assumptions in two sets, one concerning the privacy, for instance "no record devices in polling booths" or "strict management of access to ballot boxes", and the other the integrity of the election, for example "auditors will not collude with election officials" or "an effective voter registration system is used".

Helios

Finally, an Internet Voting (*i*-voting) solution (Browser only) was submitted by Ben Adida, from the Harvard University, with the so called Helios system[8]. Its development was aimed for small elections of online communities, student government, and other environments where trustworthy and secret

[7]This description regards the ballot from the Scantegrity system. The ballot from Punchscan is slightly different, but the principles are the same.

[8]More on Helios: `http://www.heliosvoting.org/`

ballots were required but coercion was not a serious threat. The web server was implemented in Django, a Python web-framework, and on the client side most operations, including those related to cryptography, were implemented with JavaScript (JS) and HyperText Markup Language (HTML) technologies. This system was inspired on [Benaloh, 2006], which in turn was based on [Sako and Kilian, 1995].

As this system was chosen to be studied and researched, a complete description of it will be given on the following sections.

4.1.2 Why Helios?

Accordingly to the statement issued by the IACR board[9], the Punchscan and Helios voting systems were picked as the finalists for the contest. But since the Punchscan team gave up, only the demo by Helios was provided. Due to the characteristics of the system, such as being completely remote and web based, the existence of documentation and technical papers, modern cryptographic protocols involved, easy testing and modification, and being open-source, this was the natural choice as the system to be further studied/used in this project.

Other systems, like Punchscan or Scratch, Click & Vote, also have good qualities, but since they are hybrid, it would complicate the testing phases. Adder and Easiest Election did not put their system's code available, which by itself has proved a major obstacle when studying them. Civitas was an interesting system, but comparing with Helios, it was at an earlier stage of development, having more technical problems. From SVIS, no information or source code was found.

Another reason to such choice, is the fact that Helios system has been used in some serious elections. The most relevant is described in [Adida et al., 2009], where the system was used to elect the president of the *Université catholique de Louvain*. A research from the Undergraduate Student Government from the Princeton University concerning reforms in elections[10], quoted Professor Olivier Pereira saying the reasons why Helios was chosen[11]:

> *Three reasons motivated this choice. First, voters have a high*

[9] Statement ("The 2010 IACR demo election"): http://www.iacr.org/elections/eVoting/

[10] A reference to that research: http://usg.princeton.edu/component/content/article/207-august-4-a-clear-direction.html

[11] Later, on the same year, Helios system turned out to be used on the Princeton Undergraduate Student Government elections.

> *level of control at each stage, and we can verify that everything is operating correctly, and thanks to the internet, a voter can verify that her vote has been recorded. This system produces the weighted count of the ballots submitted by voters, nobody (nor even a computer) can ever determine who voted for whom. Finally, the system makes it virtually impossible to vote incorrectly: first, the voter is prompted to confirm his vote, and then he has the opportunity to vote again if he thinks he is having a problem or made a mistake (that is one of the great peculiarities of the system).*

4.2 Overview

Helios is on its third version, constantly improving on the security measures and used techniques, aiming to a greater modularity, flexibility and scalability to allow different kinds of elections to different kinds of communities. In order to understand the discussions and implementations on further chapters, in this section a complete description about Helios, with emphasis in its workflow, will be given, mostly based on: the available documentation[12]; on technical papers since the first version of Helios [Adida, 2008, Adida et al., 2009]; on encountered attacks [Küsters et al., 2012, Estehghari and Desmedt, 2010]; and finally inspecting the source code and conducting tests.

To simplify the explanation of the system, we consider the following scenario: after the Camerlengo confirms the death of the Pope, the Cardinal with most power, known as the Dean, summons the College of Cardinals to elect the new Bishop of Rome (Pope). Due to high costs of travels, the Dean chooses a cheaper way to elect the new Pope by using the Helios systems, allowing the Cardinals to vote from a remote location. The adversary, or malicious entity, will be known as Cardinal Lucifer.

4.2.1 Pre-election Stage

The function of the Dean in this election is to work as the election's Administrator. Before handling election related matters, the Dean must authenticate in an Helios server, which can be made through modern mechanisms like twitter, google, facebook and even yahoo, or by registering in the system (Figure A.11, note that this authentication process is not related with the one made by the voters in a later stage of the election). Although the mechanisms are capable, if Lucifer, has access to Dean's credentials, he could impersonate him and

[12]Documentation site: http://documentation.heliosvoting.org

manage the election according to his will. Assuming that those problems will not happen, the Dean is able to create the virtual Conclave.

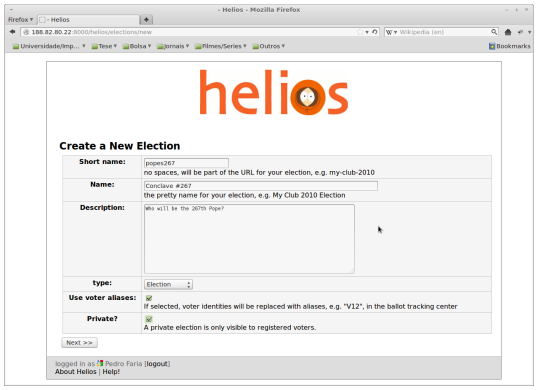


Figure 4.1: Set of parameters to create an election.

Election Details. As seen in Figure 4.1, there is a set of parameters needed to define the Conclave's general information:

- *Short name* - is the election's identification name. The name chosen will be part of the generated Uniform Resource Locator (URL) which will reference the election.
- *Name* - the official name of the election.
- *Description* - corresponds to the information about the Conclave.
- *Type* - there are two types: referendum and election. Does not have a direct influence on the server's, or election's, behaviour.
- *Use voter aliases* - using voter aliases provides an extra layer of privacy. This feature was added in [Adida et al., 2009] with two purposes:

  1. In case someone manage to decrypt the votes that are published during the election, it would still be impossible to link the vote to the corresponding voter.
  2. Some organizations may require that the voting server and authentication server are two different entities. This way, if the voting server is compromised, it would not contain any information of the election participants.

- *Private* - this option modifies the behaviour of the election in two ways. If it is private, than all the information, including the results, is confidential and it is the administrator's job to create the voters list. If it is public, the information is available to anyone, but the administrator can still decide whether the election is for anyone, or it is for only a set of people.

After defining that set of parameters, the Dean is re-directed to the home page of the election (Figure A.12). Without any special order, still remains crucial steps to completely prepare the Conclave and to finally freeze the election.

Questions. The next possible step is to create a set of questions regarding this election. Figure 4.2 exemplify how the process is done. In this case, the Dean asks the electors to choose the 267^{th} Pope and gives a list of possible answers. Then, he decides whether the final result should be relative or absolute. The last thing to decide is the number of possible answers to cast in a ballot, which in this election, the Dean allows one to three choices or a blank vote.

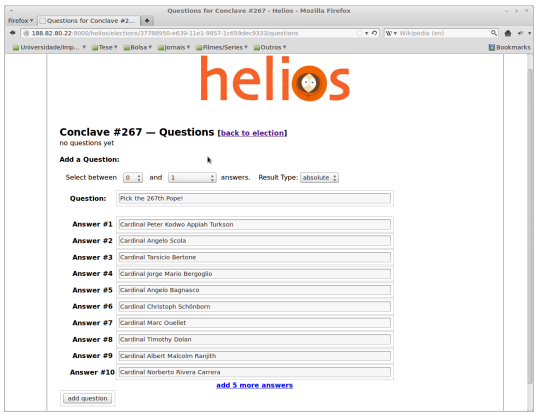


Figure 4.2: Creating questions and corresponding rules.

When comparing between the voting types defined in 2.2.1 and the possible combinations of this feature, one can conclude that there are some limitations in Helios. For instance, using Helios, the Dean could not organize elections that resort to the *Preferential voting* or the *Write-in* voting types. Later, in the next section, it will be explained why.

Voters & Ballots. After the Dean defines the crucial question about who will be the Pope, he proceeds to create a list of voters. To do so, he must

upload a file with a specific syntax as seen in Example 4.1. This file contains the list of voters in which each line corresponds to a specific voter, divided in *alias , e-mail , name*.

```
1  pedrofaria  , pedro.faria.80@gmail.com   , Pedro Faria
2  pedrolopes  , pedro_faria_22@hotmail.com , Pedro Lopes
3  vanialages  , vania.mlages@gmail.com     , Vania Lages
4  filipafaria , filipa.v.faria@gmail.com   , Filipa Faria
```

Example 4.1: Example of a list of voters

After the upload, Helios does the parsing of the file and refreshes the web page with the new data. The refreshed page will contain the names of the voters, as well as generated alias to each one and a *Smart Ballot Tracker* that will be used in the future (Figure A.13).

Trustees. In previous versions of Helios, the Dean would have to blindly trust that the system would not decrypt individual votes. Now, he is able to have more trustees (definition in Subsection 2.2.3), in addition to the system itself, using a kind of Secrete Sharing (SS), which will be explained later, in Subsection 4.3.1. However, this action of adding trustees, if used incorrectly, could jeopardize the entire election. Thus, a warning is made by Helios every time the Dean wants to add a trustee (Figure A.14).

Continuing the example, the Dean decided to add a Cardinal as a second trustee to the election. For this task, a simple e-mail with a particular generated URL and invitation to the Cardinal is sent. Then, the election will be in a kind of stationary state (as seen in Figure A.15) until the Cardinal uploads his public key or the Dean decided to remove him as a trustee.

Independent Steps. In this stage, only the Cardinal invited as a trustee will be involved. After receiving the invitation (Figure A.16), he follows the URL to the page where he will be asked to generated a new pair of keys. After generating the keys, as many times as he wants, he saves them. In this case, saving is writing the keys in JavaScript Object Notation (JSON) format on an empty file, as seen[13] in Example 4.2.

Afterwards, the Cardinal uploads the public key to the Helios server. Meanwhile, a fingerprint[14] of the public key, resultant from the function Secure Hash

[13]In this example, in order to maintain some clarity about it, the numbers were "compressed". Real-life numbers should have, in some cases, 2048 bits.

[14]Usually, the fingerprints in Helios are strings encoded in base-64.

```
{
 "public_key": {
         "g": "148874922249"..."36318763589660808533",
         "p": "163286320840"..."00238405503380545071",
         "q": "613295662483"..."23435508370458778917",
         "y": "380725418459"..."62972109814856710919"
 },
 "x": "171622450"..."632245368824456728"
}
```

Example 4.2: Example of a generated pair of keys.

Algorithm (SHA) over the key[15], is presented to the Cardinal. It is important that the Cardinal records that fingerprint because is his job to check if in fact his key is being used. He may verify the validity of the fingerprint and his secret key by using another feature of Helios (as seen in Figure A.17).

Freezing the election. Returning to the Dean's point of view, all is left is to freeze the election. This means that it will trigger a set of actions, such as:

- The election starts;
- The voters list becomes immutable;
- It is not possible to modify or create new questions;
- No trustees can be added;
- An email may be sent to every voter;
- It is now possible to cast votes.

4.2.2 Voting

After the election is prepared by the Dean, each eligible cardinal will proceed to cast his vote. From an invitation received (this election is private), he follows an URL to the virtual voting booth. The first view is a welcome page with the instructions of the voting procedure (Figure A.18), which consists in three steps:

1. *Selecting the answers* - The voter will first pick the cardinals that he believes are best suited to be the next Pope. He may select up to three cardinals, as seen in Figure 4.3, which is the number of choices limited by the Dean in the previous stage of the election.

[15]By default, the function used is SHA-256

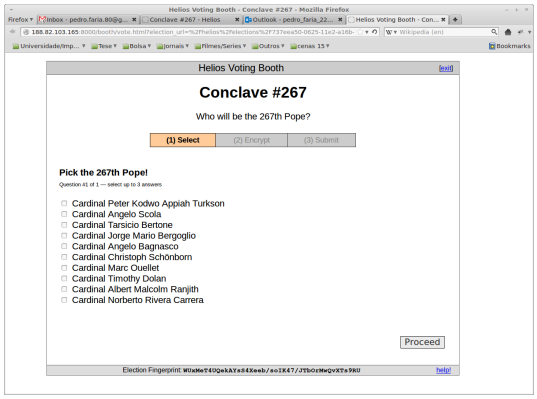


Figure 4.3: Choosing the next Pope.

2. *Encryption* - After confirming the choices, the voter's browser will proceed to the encryption of the ballot. At the end of this step, a ballot tracker, a digest of the encrypted ballot, is given to the voter (Figure A.19). Note that at this time, the voter didn't authenticated himself to Helios. In other words, anyone aware of the election could performed these operations.

3. *Submitting* - Before cast his ballot, the voter is faced with two possibilities, either he audit the encrypted ballot or he submits it. If he chose to audit, a complete description of the ballot, with plaintexts and ciphertexts, is shown to him. Then, he can use the description and a feature of Helios (or an independent one) to confirm the correctness of the encryption (Figure A.20). Afterwards, the voter is conducted back to step 2, and he could repeat this operation over and over again until he is fully convinced of the Helios' honesty. When convinced, and after a new encryption of the ballot, the voter can authenticate himself and cast his vote (Figure A.21). Afterwards, the voter can keep track of his vote by cheking the Smart Ballot Tracker (SBT) (Figure A.24).

4.2.3 Post-voting and Results

The current version of Helios removed the "end date" of the election. Thus, the Dean is in charge of ending the election. The tally is then computed by Helios, but the results will be encrypted until each trustee share is portion of the secret key(Figure A.22). At this moment, the Dean must send a warning to each one of them to upload their portion of the secret. This is a crucial stage, because, in the current version of Helios, the missing of a single trustee will completely ruin the election (the results cannot be retrieved). When all the shares are combined, the Dean is able to release the final results as shown

in Figure 4.4. An e-mail is sent to each participant announcing the results and, if there is a winner, white smoke will be visible.

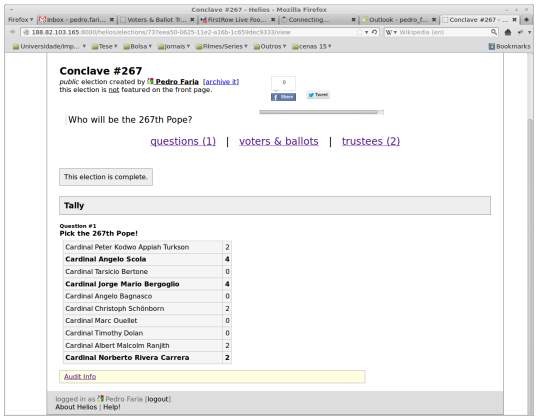


Figure 4.4: Final results.

4.3 Backstages

Before starting to explain the underlying operations performed by Helios, it is worth recalling some of its characteristics. First, although this system implements some few techniques to prevent coercion, the voters are still liable to receive threats or bribes by malicious entities. In other words, the security property "coercion-resistance" from Subsection 2.3.1 is not achieved. Another characteristic is that Helios was implemented to be fully open-audit. This means that all data stored by Helios can be retrieved through a kind of Application Programming Interface (API), and can easily be audited or verified. Helios preserves E2E verifiability, needed to assure the integrity of the elections created in this system. Finally, Helios uses Homomorphic Encryption (HE) as the main cryptographic technique to ensure the security of this voting system.

This section will follow the same structure as the previous section. For each stage, a description of the techniques used will be given, with emphasis on the cryptographic techniques used.

4.3.1 Pre-election Constructions

As it was mentioned in Subsection 4.2.1, after the Dean freezes the election, all the parameters must be defined. Using the API, those parameters will result in JSON object of Example 4.3. The `<PUBLIC_KEY>` and `<QUESTION>`

tags corresponds to other JSON structures that will be explained later. The first step to audit the election is to confirm that the message digest of that structure is the same as the one presented on the website. This will assure the integrity of the public key and the other election parameters involved on the encryption of ballots.

```
{
  "cast_url": "http://188.82.103.165:8000/helios/elections/737
     eea50-0625-11e2-a16b-1c659dec9333/cast",
  "description": "Who will be the 267th Pope?",
  "frozen_at": "2012-09-24 09:03:52.125605",
  "name": "Conclave #267",
  "openreg": false,
  "public_key": <PUBLIC_KEY>,
  "questions": [<QUESTION>],
  "short_name": "popenumber267",
  "use_voter_aliases": true,
  "uuid": "737eea50-0625-11e2-a16b-1c659dec9333",
  "voters_hash": null,
  "voting_ends_at": null,
  "voting_starts_at": null
}
```

Example 4.3: Election structure

Key Generation. In order to create the public key pk of the election, each trustee, together with Helios, will generate an ElGamal Encryption (ElGamal) public key pk_i, which will be composed by the prime p, the prime-order q, the generator g and the public value y_i. The value x of $y = g^x \mathsf{mod} p$ will be secretly saved by each trustee. Afterwards, all pk_i will be combined, multiplying the public values y_i. The result, pk, will be stored in Helios server as the JSON structure in Example 4.4 [16].

```
{
  "g": "0966798589660808533",
  "p": "15036974934746820 71",
  "q": "78917",
  "y": "7865248494343439092"
}
```

Example 4.4: ElGamal public key structure

[16] Once again, the numbers were shortened for ease of understanding of the example. For instance, the value y is a 2048 bit long prime number.

Questions. At this point, the Dean has only to define the questions of the election. As it was mentioned before, Helios uses HE to tally the final result of the election. Thus, the voting types implemented by Helios were created in order to make it possible to use the homomorphic properties in the tallying stage (see Section 3.1). This is the reason why the Dean was not able to chose a different kind of voting back in Subsection 4.2.1. The structure from Example 4.5 was generated according to the question created by the Dean.

```
{
  "answer_urls": [null, null, null, null, null, null, null,
     null, null, null],
  "answers": ["Cardinal Peter Kodwo Appiah Turkson",
    "Cardinal Angelo Scola",
    "Cardinal Tarsicio Bertone",
    "Cardinal Jorge Mario Bergoglio",
    "Cardinal Angelo Bagnasco",
    "Cardinal Christoph Sch\u00f6nborn",
    "Cardinal Marc Ouellet",
    "Cardinal Timothy Dolan",
    "Cardinal Albert Malcolm Ranjith",
    "Cardinal Norberto Rivera Carrera"],
  "choice_type": "approval",
  "max": 3,
  "min": 0,
  "question": "Pick the 267th Pope!",
  "result_type": "absolute",
  "short_name": "Pick the 267th Pope!",
  "tally_type": "homomorphic"
}
```

Example 4.5: Question structure

4.3.2 Encrypting Ballots

The encryption of each ballot is made locally in the voter's browser. No communication during this stage is supposed to happen. Another requirement is that the access to the virtual booth should be anonymous[17] in order to prevent some attacks where the knowledge of the voter is required, such as impersonation. In fact, the only time the voter should present some of his information is at the step of the vote's submission, where the only available information is the encrypted vote.

[17]This does not happen in Helios' private elections.

Each vote structure saved in Helios is composed by several encrypted answers, the election fingerprint and the election ID, as shown in Example 4.6. According to the documentation, the only purpose of the ID is for convenience when generating some HTML pages. On the other hand, the fingerprint is important because it will link the vote to the election. Thereby, each encrypted answer has been formed according to this fingerprint, which means that when verifying the votes, one can compare the fingerprints and see, for instance, if no question or choice has been changed. The final attribute of this structure is an array of encrypted answers, one for each question of the election.

```
{
  "answers": [<ENCRYPTED_ANSWER>],
  "election_hash": "WUxMeT4UQekAYsS4Xeeb/soIK47/JTbOrMwQvXTs9RU
     ",
  "election_uuid": "737eea50-0625-11e2-a16b-1c659dec9333"
},
```

Example 4.6: Vote structure

The encrypted answer structure is far more complex, as can be seen in Example 4.7, which is the result of an encrypted answer from the previous example. This structure has three attributes:

- *choices* - is an array in which the number of elements is equal to the number of choices of each question. Each element is an Exponential ElGamal Encryption (EEG) ciphertext, composed by an "alpha" and "beta", which respectively corresponds to c_1 and c_2 of the EEG description in Subsection 3.1.2. The ciphertext was created with the election's public key and a particular randomness for each element.

```
{
  "choices": [
    {"alpha": "1454391040", "beta": "9197231515"},
    {"alpha": "8983218606", "beta": "1826285502"},
    ...
    {"alpha": "9746932504", "beta": "9780504308"}
  ],
  <Individual_Proofs>
  <Overall_Proof>
}
```

Example 4.7: Encrypted answer structure

- *individual proofs* - demonstrated in Example 4.8, this attribute is an array that also contain the same number of elements as the previous

attribute. Each element contains disjunctive ZKP that the respective element in *choices* encrypts either a 0 or a 1, i.e. prevents a malicious voter to encrypt a value bigger than one, which in turn could cause a tremendous impact on the final result, as will be explained later. The ZKP involved is the Chaum-Pedersen protocol described in Subsection 3.4.3. Comparing with Figure 3.6, the attribute "challenge" is c, the "response" is t, and the "commitment", which contains "A" and "B", is the pair (g^s, y^s).

- *overall proofs* - This is also an array, but the number of elements is equal to the maximum number of possible choices, plus one. For instance, recalling the previous example, each cardinal could choose at least three candidates to be the next Pope. For each number of possible choices, $\{0, 1, 2, 3\}$, an ZKP will be generated, making, in this case, a total of four. The ZKP used is the same as before. This proof will guarantee the cardinal did not chose more than the limit set by the Dean.

```
"individual_proofs": [
  [{"challenge": "332889775",
    "commitment": {"A": "1055566340", "B": "105207753264"},
    "response": "47706"},
   {"challenge": "431463304",
    "commitment": {"A": "2837462863", "B": "142091740803"},
    "response": "18979"}],
   ...
],
"overall_proof":[
  {"challenge": "212991023",
  "commitment": {"A": "6312567672", "B": "782646636317"},
  "response": "48028"},
  ...
]
```

Example 4.8: Individual and Overall Proofs

Auditing. Recall that the voter in the voting stage could choose to inspect the validity of the formed encryption. This time, the structure produced will have the information of the structure in Example 4.6 plus all the randomnesses used to encrypt the answers. This way, a voter can verify has many times he wants if the encryptions are well formed and whether in fact it corresponds to his choices. To prevent the verified encrypted ballot could be used has a receipt to vote selling, this vote will become useless and impossible to cast. On the other hand, it could be posted in a Bulletin Board (BB) for public checking.

4.3.3 Tallying and Decryption

To compute the tally, the Helios server will simply put in use the additive homomorphism of each EEG ciphertext. As demonstrated in Subsection 3.1.2, by multiplying all the ciphertexts corresponding to a candidate, the result will be the encryption of the sum of all the votes. For instance, in the general example, by multiplying the first element of the attribute answer of each casted vote, the result will be number of cardinals that vote for the candidate "Peter Turkson", as can be seen in Figure 4.3. This way the final result can be computed without revealing a single vote.

Decrypting the tally implies having the knowledge of the secret key that corresponds to election's public key used to encrypt the votes. Thus, after computing the tally, all the trustees involved must upload their secret key in order to combine them and thereby create the election's secret key, so that is possible to decrypt the tally.

Auditing. With the API of Helios, all the public data of the election is available to public. Helios also provides one tool to check the integrity of that data, assuring the voters that even if the administrators were corrupt, they could not produce a fake tally. The tool takes as input the election URL and using the API will retrieve all the information regarding (Figure A.23):

- Election - check the parameters of the election, comparing with the public fingerprint, and all the voters who took part in the election.
- Ballots - for each voter, the tool will download the last casted vote. Then, for every question, will check for the correctness of the options and the global values, making use of the ZKP presented in 4.3.2.
- Trustees - to verify if the public key was well formed and if the trustees presented are in fact the trustees supposed to protect the election.
- Tally - to inspect the vote count and if the encryption was made using the correct key.
- Final Result - compare the final result published with the final result inspected.

4.4 Assumptions & Improvements

The number one rule when creating an *e*-voting scheme is that it is impossible to create a secure system without making a few assumptions. For instance,

it is impossible to create a completely remote e-voting scheme in order to guarantee coercion-resistance without compromising the rest of the security properties (see Subsection 2.3.1). In fact, to assure coercion-resistance, at least the assumption "the voter must have a single moment of privacy" is required. This section is divided in two: the first part will describe the assumptions made by Helios; and the second part will present some works and improvements over Helios, in order to change its assumptions or add new capabilities or features.

4.4.1 Assumptions

The first motto of Helios was "trust Helios for privacy, trust no one for integrity". This principle still remains on its third version and it is the base for the most important assumption when using Helios as a secure voting system: trust Helios. Trusting Helios assumes that the server is not corrupted, otherwise the following problems could occur:

- Single ballot decryption - as described in Subsection 4.2.3, in order to decrypt the final tally, all the trustees must upload their private knowledge of the election's secret key, so Helios is able to do it. In the end, Helios will have all the encrypted votes, the links to its owners and the secret key. So no measure exists that prevents a corrupt Helios to decrypt those single votes. Note that the secret key to decrypt a single vote is the same as the secret key to decrypt the tally.

- Ballot stuffing - Figure 4.5 shows the election's state moments before the end. As can be seen, four out of nine voters did not cast their votes at that moment, and two of them will not cast it ever. This opens a issue in case of a corrupt Helios, because it could impersonate inactive voters and cast their votes at free will and no one, except those voters, would find out.

Safe computer. Recalling now the voting stage in Subsection 4.2.2. The operation to cast a vote is done through the browser of a voter's personal computer. Thus, the privacy of the voter is directly related to that browser/-computer's integrity, which in turn is prone to virus, worms, spyware or any other kind of malware. Therefore, another assumption is required in Helios: all voters cast their votes in uncorrupted browsers/computers. In the next chapter, an attack regarding this assumption will be explained.

Computational Privacy. As it was explained before, the choices are encrypted using EEG (see Subsection 3.1.2). Like most of other cryptographic

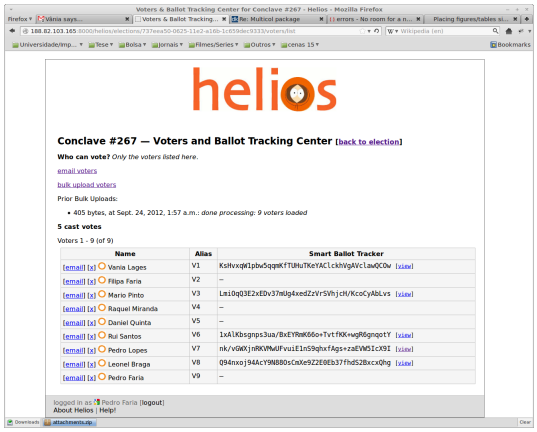


Figure 4.5: Moments before the election is finished.

techniques, its defense is based on a known hard problem, which in this case is the Discrete Logarithm Problem (DLP). This means the protection of using EEG does not result in everlasting privacy. Depending on the values involved, it might take decades or centuries to decrypt it by using means of cryptanalysis.

4.4.2 Improvements

Since Helios is being actively studied by the academic and scientific community, it is normal that some new suggestions, or even implementations, arise in order to improve this system. The following paragraphs briefly describe recent works, some of which are still in progress, that try to solve some of the Helios problems.

Usability. In [Karayumak et al., 2011] and [Karayumak, 2010], studies on usability had been made[18]. Some problems related to the site's aspect were pointed, such as a mix of links and buttons, inconsisting wording when describing each step of the voting stage or the fact that there is no back buttons. Another lack of guidance and consistence was found when changing between the steps of re-encrypting the ballot and verifying its encryption in the voting stage and auditing the election. According to their research, the steps were too confuse and error prone, which cause most of voters to skip those steps. To see the importance of a good usability in a election, if most voters do not verify their votes, the election's integrity can not be assured. This problem is considered a major con argument against E2E verifiable *e*-voting schemes.

[18] Other usability tests performed can be found in: `http://www.jannaweber.com/?p=129`.

Provable ballot privacy. An attack against Helios has been found and demonstrated in [Cortier and Smyth, 2011] and more specifically in [Smyth, 2012]. This attack is known as the "Replay Attack" and consists in copying casted votes. Imagine the following scenario: in an election with three voters, Alice, Bob and Mallory. Mallory wants to know who is the chosen candidate by Alice. Since Alice will not reveal her vote, Mallory copies the casted vote of Alice. In the end, Mallory will know that the candidate that receives two or three votes will be the candidate chosen by Alice. But, as stated in the Helios documentation[19], this attack only matters if a great number of voters are willing to give up their votes in order to invade the privacy of a single voter.

Nevertheless, some counter measures had been created to repel this attack. The first suggested was to delete duplicated fingerprints in the smart ballot tracker (remember Figure 4.5). This measure has the setback of being easily overcome because since the fingerprint is computed over a JSON structure (Example 4.7), it can be corrupted, for instance, by inserting an extra space. Hence, the value of the fingerprint will be different but the cryptograms will not. Currently, Helios is adopting more complex measures to detect copies and sub-copies of encrypted ballots. In [Bernhard et al., 2011], a more formal approach to solve this problem is presented. Basically, they adapt known algorithms that prevents a special attack, known in literature as "chosen ciphertexts attack", to the Helios *e*-voting scheme. They also modify the ZKP in order to prove the correction of the encryption.

Mixnet with Helios. As mentioned before, one limitation of Helios was the fact that only a few voting types were permitted. Another flaw was the creation of a ciphertext per candidate in the ballot encryption. As a result, ballot encryptions for a great number of candidates would translate in a great number of randomnesses, ciphertexts, individual proofs and overall proofs. Thus, in [Bulens et al., 2011], the authors go back to the first philosophy followed by Helios, the use of mixnet *e*-voting schemes, and create a variant of this system.

Apart from the cryptographic techniques involved, there were significant changes on the Helios workflow. The major difference is the two kind of trustees, the shuffling trustees and the decryption trustees, which consequently changed how the generation of the election parameters were performed as well as the decryption stage. Moreover, they put in use a secure submission augmented cryptosystem in order to avoid some kind of replay attacks. In the lessons learned from this variant of Helios, the authors give emphasis to the

[19] More precisely in: http://documentation.heliosvoting.org/attacks-and-defenses.

fact that HE is still a better choice if there is a small number of candidates. Otherwise, mixnet has a better performance and has the advantage of providing a wider set of voting types. Nevertheless, a future plan already mentioned in the official site of Helios, is to incorporate this feature in future versions.

Everlasting privacy. Still not published, the ongoing work by Jeroen van de Graaf et al.[20] was presented in SecVote2012 summer school. This work tries to overcome the computational assumptions regarding privacy property. According to them, it is preferable to have an unconditional privacy of the ballots with an computational integrity of the tally than the opposite. In the motivational examples, they show the case where after decades of trying, an evil dictator gets elected democratically. Then, after years of cryptanalysis, he decrypts the election ballots where he lost and goes after the voters who voted against him. Regarding Helios, this is true for its first version, where the name of the voter was placed next to the ballot submitted in the BB.

The basic idea is to use Pedersen commitments as an alternative way to encode the votes. The use of HE still remains but with a few changes. Some secret channels are needed to provide secret knowledge in order to decrypt the votes, with the use of the Paillier encryption (see appendix A.2.2). However, some assumptions still exists, such as "the Paillier encryption is semantically secure". As it was mentioned above, this is an ongoing work, and it still does not have a practical implementation.

4.5 Summary

The first section of this chapter presented a contest which aimed to create a secure remote *e*-voting system. It is interesting to compare the requirements of the contest with the requirements found and studied in chapter 2. One can easily conclude that the security of a *e*-voting system can not rely only in cryptographic matters. Nevertheless, they are essential to provide security.

A remote *e*-voting system - Helios - is presented in detailed. This system is not perfect, but is one of the most prized and researched that can be found in literature. Its principal characteristics: be E2E verifiable, which will protect the election's integrity, and oriented to low coercion elections, i.e. this system was implemented for elections where coercion is not a relevant issue. Of course, due to that fact, this system could not be used for high stake elections, such

[20] Presentation: `http://secvote.uni.lu/slides/jvdgraaf-everlastpriv.pdf`.

as government elections.

Finally, the assumptions that guarantees the security of Helios, along with works that aim to improve it, can be found in the last section. The next chapter will be about other improvements and ideas on this system.

Chapter 5

Improvements

> ***Eddy**: I assume he will still be carrying when he comes back from the job.*
> ***Soap**: Oh, you assume, do ya? What do they say about assumption being the brother of all f*ck-ups?*
> ***Tom**: It's the mother of f*ck-ups, stupid!*
>
> *Lock, stock and two smokin' barrels*

Up to this point, Chapter 2 gave a detailed overview of the Electronic Voting (*e*-voting) world. Next, in Chapter 3, the majority of cryptographic primitives used in *e*-voting protocols were presented, to achieve the requirements and security properties of the previous chapter. Finally, Chapter 4 shows how those primitives were employed in a successful *e*-voting system called Helios, an End-to-End (E2E) open-audit system, whose biggest goal is to protect an election's integrity. In this chapter, we put into practice the knowledge acquired during the study presented in the previous chapters. Our goal is to create or modify features of Helios in order to improve security and usability of this system. Thus, the following sections will describe some changes made to the system and what results are expected from them. In:

- Section 5.1: presents how a double-password scheme could increase protection against coercion and vote selling.
- Section 5.2: a new entity is introduced to decrease the level of trust in Helios and provide anonymity when a voter accesses the virtual booth.
- Section 5.3: employs a cryptographic primitive to deal with a problem found with the above solution.
- Section 5.4: using a technique called code signing, more guarantees of authenticity and integrity are added to the voting booth.

- Section 5.5: presents a prototype of an Android application, which aims to be a new kind of voting booth.

Finally, Section 5.6 makes a brief summary of this chapter and mention some other ongoing work.

5.1 Working against coercion

As it was mentioned throughout the previous chapter, Helios was not conceived to achieve coercion-resistance. This section presents measures to fight coercion in Helios. It is divided in two parts: starts by describing simple attacks and available counter-measures to protect voters. Then, in Subsection 5.1.2, a solution is proposed, based on changes to the Helios workflow. We explain how the solution works, describing some details of implementation and finally discuss the implications of the proposed solution.

5.1.1 The problem

The following paragraphs describe situations in Helios that may lead to terrible problems in elections that use this system.

- *Vote buying* - continuing with the example of Section 4.2, imagine that Lucifer wants to buy some votes to support his favorite candidate. To do it, he would approach some cardinal and offer a bribe if that cardinal proves to vote in his candidate. In turn, the cardinal would cast his vote, i.e. he would follow the steps described in Subsection 4.2.2, right in front of Lucifer. If the resultant fingerprint maintains until the end of the election, in other words if it appears in the Smart Ballot Tracker (SBT), Lucifer will know that the cardinal voted in his favorite candidate and he will give him the bribe. Note that Lucifer can approach as many voters he wants, in order to corrupt them, because he only needs a small period of time during the available time to cast a vote, as shown in Figure 5.1 *a*).
- *Threats and bullying.* Now Lucifer intimidates a certain cardinal to vote in a particular way. Like the previous situation, he can approach at any time one of the voters and coerce him. Helios may already provide measures to fight low coercion, but for clarity, consider the following two levels:
  - *Bullying* - if Lucifer only coerces a cardinal during a small period of time during the election, as seen Figure 5.1 *a*), that cardinal still

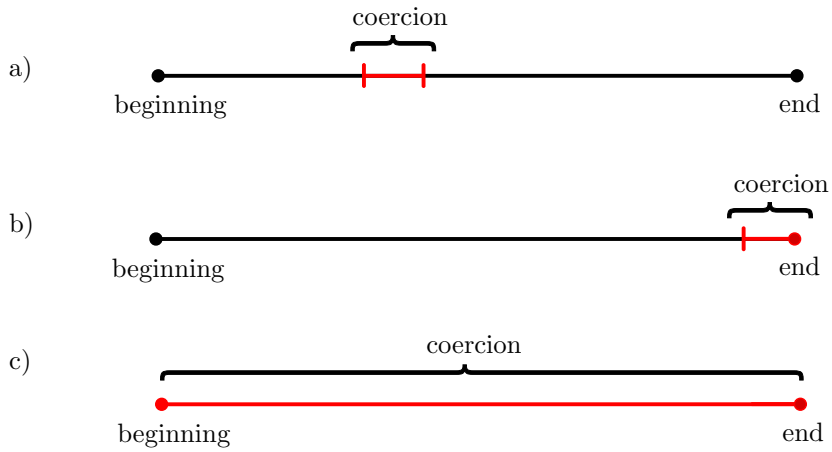


Figure 5.1: Different periods of coercion time.

has the opportunity to re-vote and change the casted vote at a later time. But Lucifer can still be succeed if he coerces the cardinal during all the time of the election, as seen if Figure 5.1 *c*), or in the last momment *b*). Note that the original Helios will always update the SBT.

– *Threats* - considering a higher level of coercion, if Lucifer threats a cardinal, he only needs to make sure that the cardinal will not change the vote until the election ends, just like the vote buying situation.

5.1.2 One possible solution

There are several ways suggested in literature to provide coercion-resistance in *e*-voting schemes. The solution we have developed to Helios was inspired in [Clark and Hengartner, 2011]. To overcome the problem we modify the workflow of the system, by creating a registration stage before the election is created. Like most improvements in Helios, this solution has also assumptions, namely:

1. We continue to trust Helios for privacy
2. The registration is made in a secure way, e.g. in person.
3. When the voter is alone, he will seize the opportunity to cast a vote.

Continuing with the general example, each cardinal is invited to register in Helios. This registration will be independent of any election and the cardinal must fill a typical form with his information and a set special passwords. This

set of passwords will be composed by one called *secured pass* and a *coerced pass*, and all of them should be easy to remember, specially in stressed situations. Helios will then store the passwords in a secure way, for instance, by saving their fingerprints instead of the real value. This stage will end before the election starts. Afterwards, it will proceed normally with the Dean preparing the election and inviting all cardinals to participate. When each cardinal access the voting booth and proceed to cast his vote, instead of credentials he will be asked to insert one password. Helios will count only the votes submitted with the *secured pass*, hence the need to trust Helios.

How it will affect coercion? First, this approach will affect Helios' behaviour, depending on the type of password submitted. Note that it is intended to modify the Helios to a minimum in order to not remove the features of its original version, so the voting booth will have no differences. Nevertheless, fooling the coercer requires changes in the SBT, as it will be explained accordingly to the following situations:

- *No coercion* - the cardinal simply inserts the *secured pass* when it is asked to cast the vote. In the SBT, the fingerprint will be normally placed next to the cardinal's alias. If the cardinal do not insert an valid password, Helios will not warn him, but the ballot will be ignored. If the cardinal re-votes with the *secured pass*, Helios will refresh the SBT.

- *Minimum period* - during the time of the election, Lucifer will approach a cardinal and coerce him to vote in a certain way. When the password is asked, the cardinal will submit the *coerced pass* and cast the encrypted ballot. Helios is now warned that the cardinal is under threat, so it will update the smart ballot tracker to apparently accept the ballot. Since this is a small time period, when Lucifer leaves to coerce another voter, the cardinal may re-vote in a desired way. Helios will not update the SBT with the valid vote, which makes Lucifer uncertain whether the cardinal has changed his vote or not. In case of a second coercion attack, the cardinal submits again an encrypted ballot with the *coerced pass*. In other words, after the first submission with the *coerced pass*, Helios will only update the SBT if the ballot is submitted using that password.

- *Last moment* - this solution still solves the coercion problem if Lucifer coerces the cardinal moments before the election ends. But to solve it, a weak assumption is needed: during the free time, the voter cast a valid vote (assumption 3). If the cardinal casts one valid vote prior to the coercion time it will still be counted when submitting a ballot with the *coerced pass*. Helios will refresh the SBT only to fool Lucifer.

- *Maximum period* - this situation results in the same problem as if the previous assumption fails. This solution cannot solve it. If the coercion time is the same as the duration of the election, the cardinal cannot fool Lucifer and vote in the way he wants. On the other hand, this solution still brings advantages when comparing with the original Helios. Like the cardinal, Lucifer cannot has his vote convincingly submitted because, like the previous situations, he does not have the guarantees that the encrypted ballot was submitted using the *secured pass*. Most likely, this scenario will end in abstention by the cardinal, with Helios knowing why. Another benefit of this solution when comparing with the original version, is that one coercer can only successfully make one voter to abstain, since he needs to spend all the duration of the election with that voter.

Implementation details. The first problem regards the registration stage. Instead of following the second assumption, it was decided to keep the invitation by e-mail, for the sake of clarity. This invitation will lead to a registration form where the voter will choose the two kind of passwords (Figure 5.2).

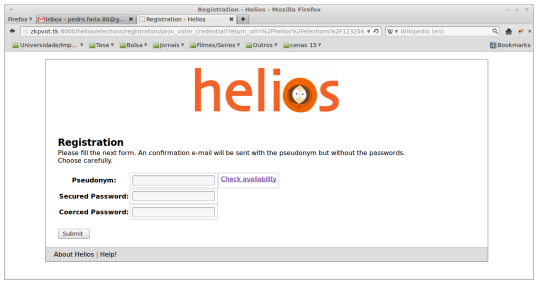


Figure 5.2: Registration with two passwords.

Internally, the voter structure was improved to receive two more attributes, the *coerced pass* and *secured pass*. Only the digest of those passwords are stored. Together with an instance of an Helios' election, a new JavaScript Object Notation (JSON) structure is formed, as seen in Example 5.1.

Visually, no alterations were needed. This was a naive implementation, only to be considered as a proof of concept. There were other personal information, regarding the voter, ready to be stored, such as phone number, in order to help authorities deal with coercing situations. But this required a more complex approach to this problem, therefore it was left for future work.

Online registration. The registration was conceived to be in-person, but is interesting to see how the e-mail feature could complicate the attack of a

```
{
 "election_uuid": "90ae293e-f94a-11e1-b06d-12313f028a58",
 "name": "Leonel Braga",
 "uuid": "53f3df48-b57f-4f3f-9eed-a810efe018d2",
 "voter_id_hash": "L5lHLWFz6rlMWhxCxNRDIKkz7K6lO6CKXH5z+OnINMc"
    ,
 "voter_type": "password",
 "secured_pass": "soQYAIT5bLwKIYfLrECOAioaoPQg7o6LjjBCu0b6yg8",
 "coerced_pass": "hrkV/MYdFMp+hynqiiMTk32oBVZ/Xzu5nhs2m9m6HnI"
}
```

Example 5.1: New voter structure jointly with election information.

coercer. He would need to spend at least the last moment of the registration stage until the start of the election, assuming that the voter will complain to the authorities. To better see the impact of what this implies, imagine that the ellipsis of Figure 5.3 means weeks, i.e. the time between the end of the registration stage and the start of the election. Moreover, this time could be extended if there were means to fool the coercer in the registration stage. Nevertheless, the solution makes coercion impractical, or at least too costly.

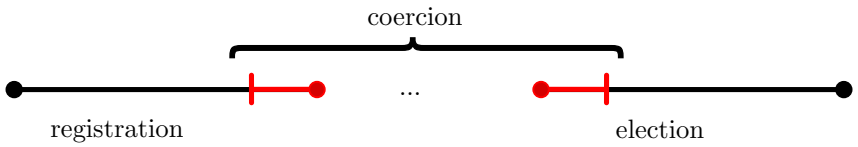


Figure 5.3: Amount of time needed to successfully coerce a voter.

Of course, for a system be considered secure, all scenarios, even the impracticable ones, must be taken into account. But in case of large elections, for instance government elections, a few coerced voters would not make much of a difference.

Con arguments and open problems. There is a longstanding battle between the authors who defend that the most important security property that a system must preserve is verifiability and the authors who defend complete privacy. Apparently, when designing an *e*-voting scheme, achieving one of the properties implies making the other very hard to obtain. This is precisely what is happening with the solution proposed. If on one hand the changes improved the defenses against coercion, on the other they may jeopardize the verifiability of the election. More specifically, it is still possible to check individual encryptions for correctness, but in turn, verifying the tally cannot be made by the coercer. That is, the coercer cannot know which votes are being counted

since he is able to calculate the receipts of the SBT and check for his coerced vote. Summing up, the E2E universal verifiability (see Subsection 2.3.1) is damaged by the presented solution.

Hence, a big open problem is how to achieve the damaged property and still preserve the anti-coercion measures. The assumption to leave the integrity of the tally to the hands of Helios is too strong. Adding an external authority in charge of checking the integrity lead to the usual problems of collusion. Removing the receipts, i.e. remove the fingerprints of the ballots, may lead to a solution for universal verifiability but harms the individual verifiability property. Hence, one can conclude that improving coercion-resistance in Helios has no trivial solution without compromising the rest of security properties.

Another con argument may be the poor usability of this solution. It is required that the voter memorises two different passwords and Helios cannot warn him if he accidentally uses a bad one. This goes against the definition of usability from Subsection 2.3.2, where it claims that the interface should help the voter from accidentally cast wrong votes.

5.2 Pseudonyms

When studying Helios, a contradiction in the voting stage was found. In the pre-election stage (Subsection 4.2.1), the Dean may choose to create a private election. By doing so, it prevents external entities to access the election's home page and the voting booth, and thus protects all the data related to that election. In order to eligible voters access the booth, they must insert the same credentials that they use to later cast a ballot. But accordingly to the documentation, the access to a booth should be anonymous, to prevent a corrupt Helios from giving a tampered booth to a targeted voter. Recall that for that voter, the biggest defense against a tampered booth was that the authentication was made only after the ballot encryption, which does not prevail in private elections.

This section presents a possible solution to defend targeted voters and tries to prevent the problems that may happen with a corrupted Helios. Subsection 5.2.1 explains the need of registration to achieve anonymity and how it was used to overcome a different problem. The next subsection propose a solution with similarities with the previous one, but with a few changes that may decrease the level of trust on the entities involved.

5.2.1 The need of registration

The registration problem has already been approached in [Adida et al., 2009], but with a different motivation. According to the authors, their motivation was the potential risk of ballot stuffing, especially considering balanced elections. Since no registration is needed on the original Helios, inactive voters can have their votes submitted by a corrupted version of Helios. The authors assume that if a voter wants to register in an election, then he will be an active part of the election and will get a permission to vote. Hence, there will be no credentials to inactive voters.

Since their solution was specific for one case, the university election, they used the university's authentication infrastructure to vote on behalf of voters, as seen in Figure 5.4. Thus, this new entity must be trusted, since he is in charge of submitting ballots. There is no detailed description: of how it was implemented, besides the use of oAuth protocol[1]; of the data exchanged in the communications between the entities; about the registration process and how it was conceived.

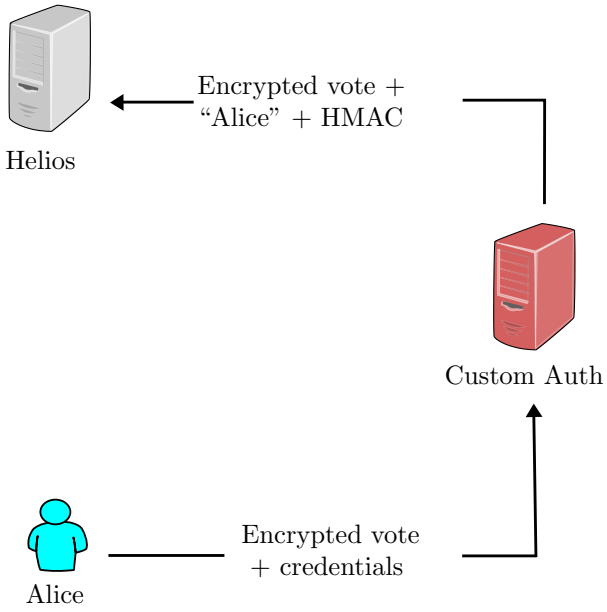


Figure 5.4: New entity introduced in Helios.

Registration. According to the authors, every potential voter receives an e-mail with a link to a secure registration website. Then, the voter authenticates with his university credentials. A random alias is assigned to him, along with a

[1]Protocol description can be found in http://oauth.net/

password. A document signed by the university's central authority is given to the voter as a proof of registration. Finally, the voter is registered and is able to cast votes on the election. This process gives a lot of power to the university's central authority. It centralizes the knowledge of the alias, passwords, and IDs of the voters. It can impersonate any voter, at any time during the election. A trust assumption is needed in order to prove the security of this scheme. Moreover, the contradiction still maintains. The access to a private booth is done using the same credentials that are used to cast a vote.

5.2.2 Independent entities

The solution proposed in this subsection consists in creating an extra entity independent of Helios, and it will be described comparing with the workflow presented in 4.2. The new entity will be called Registration Authority (RA), as seen in Figure 5.5, and is responsible for managing pseudonyms and passwords among the voters.

Pre-election Stage. Recall when the Dean uploads a list containing all the cardinals supposed to participate in the election (Example 4.1). Then, Helios sends a list of valid tokens to RA, and for voters, instead of sending an e-mail with credentials to enter in the election, Helios will send a link for the virtual booth, which is the same for every voter, a registration link with a particular token t_i and an end date for the registration process. Helios will then use a mechanism of Public-key Cryptography (PKC), say ElGamal Encryption (ElGamal), and publish a public key for the users encrypt their future pseudonym. Note that this key has nothing to do with the key that encrypts the ballot.

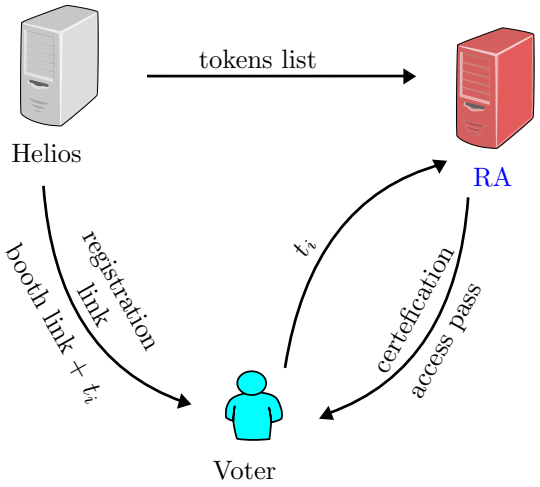


Figure 5.5: Registration stage.

Afterwards, each voter will use the token t_i received and use it to register with the RA. The token will act as an ID to prevent multiple registrations by one voter and as an authorization to get the general password to access the booth . Throughout the registration, the voter will choose his pseudonym that encrypts with the Helios public key, and the vote casting password. In the end, a certificate signed by the RA must be returned to the voter, to act as a proof of registration, along with the above mentioned access password.

Freezing the election will also have additional actions when comparing to the original Helios. The system will first interact with the RA, by requesting all the voters who registered with the RA that used valid tokens and the generated password to permit accesses to the booth. The RA returns the list of encrypted pseudonyms and respective passwords, but without the tokens used to register, along with access password. This action will be the last of the RA on this election, as shown in Figure 5.6. Helios will end this stage by publishing the list of the encrypted pseudonyms that are authorized to cast a ballot.

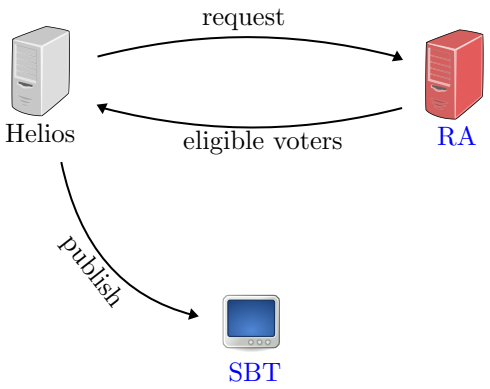


Figure 5.6: Publishing electors list.

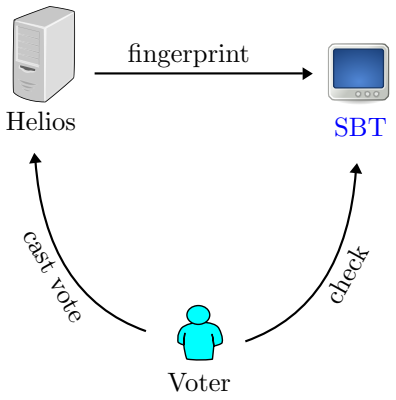


Figure 5.7: Voting.

Voting. Every voter that registered in the election may now verify if his encrypted pseudonym is published on the SBT. If it is not, he has a proof that he is an authorized voter by presenting the certificate returned by the RA. He then accesses, anonymously, the voting booth with the access password given by RA. Since this password is the same for everyone, Helios cannot know who entered the booth. To cast a vote, the voter will authenticate using the decrypted pseudonym. Figure 5.7 sums the voting process.

A new verifying stage. The remaining election will be running the same way as if it happened using the original Helios. The trustees, tally and decryption do not have any difference. But after the end of the election, a new verifying stage is needed to prevent Helios from creating fictional voters. Thus, after the voting, RA must check that the number of certificates he produced is greater or equal to the number of casted votes. In other words, Helios cannot produced valid credentials without the risk of being found out.

Assumptions. Due to the biggest assumption of the original Helios, that one must trust the system to not decrypt single ballots and thus violate the voter's privacy, it was decided that the solution must be designed in order to decrease the level of trust on Helios. Hence, the chosen approach was to create a new entity, RA, in charge of "half" the responsibilities of Helios. Nevertheless, a crucial assumption is needed in order to prevent that a fake election fools the voters: no collusion between Helios and RA. It is almost equivalent to say that at least one of those entities is honest. Still, it is a much weaker assumption than "Trust Helios". Moreover, this solution has several advantages when comparing with the original Helios and it is prepared for some individual corruptions, for instance:

- Unlike the original, if this Helios is corrupt and decrypts single ballots, it will not be able to say who voted for whom. Moreover, he cannot link voter to pseudonym, and therefore voter to ballot.
- Unlike the original, a corrupt Helios cannot say who is accessing the private booth.
- Even if RA finds the access token to enter the private booth, he is still not able to impersonate voters since he has no way to link decrypted pseudonyms to passwords during the available time to cast ballots.
- A corrupt Helios cannot add new pseudonyms with a vigilant RA.
- Since the registration is made by active voters, ballot stuffing by Helios has high probability of being discovered.
- Helios cannot risk to impersonate a voter during the registration stage since he has no way to figure out if the voter is active or not.

Credential selling and open problem. If vote selling during the voting stage can be prevented using the technique from Section 5.1, there are some problems regarding the credential selling. Since the access to the registration site is anonymous, the RA has no way to know if the person is authentic or not.

The only possible way to ensure the authenticity of that person is registration in-person. Even using smartcards or other kind of technology, there is no way to remotely guarantee authenticity. Moreover, in this solution there is no way to prevent RA from creating fictional voters. He may have problems with accessing the booth, but it would imply that every voter in the election must be honest. In the next section, a solution to this problem is proposed.

5.3 Corrupted RA

The previous solution leaves an open problem: since RA receives all tokens of registration, how to prevent that, at the end of the registration stage, a corrupted RA do not use the unclaimed credentials for self gain. For instance, it could create several pseudonyms that Helios would not discover if they were valid or not. Moreover, even if the access to the virtual booth is prevented, RA could sell the unused credentials to some corupted voter. In this section, a solution is presented using a kind of Zero-Knowledge Proof (ZKP).

5.3.1 Zero Knowledge Sets

To find a solution to this problem, it helped detecting more precisely where is the flaw. Remember that RA is supposed to only give the access booth password to eligible voters. Hence, he needs to identify the valid tokens. On the other hand, because he has the set of valid tokens, he is able to produce fake credentials. Summing up, we want a way for RA being able to verify if a token belongs to the set of valid tokens. The answer was found in [Micali et al., 2003], where the authors constructed a ZKP for membership, or not, to a set, which they called Zero-Knowledge Sets (ZKS).

Recall ZKP from Subsection 3.4.3. As detailed there, these kind of proofs are between a Prover and Verifier, where in this case, the Prover wishes to prove to Verifier whether an element belongs to a set or no, in a way that achieves *soundness*, *completness* and *zero-knowledge*. Since the construction of the ZKS proposed in [Micali et al., 2003] deals with complex schemes, such as Merkel's authentication trees, that fall outside the scope of this thesis, it was decided to exemplify it using another construction found in literature. More precisely we will consider the protocol presented in [Xue et al., 2008], which is proven secure in the "random oracle model" and "strong RSA assumption". This construction is composed in four stages: setup, commit, prove, verify. Figure 5.8 sums how the protocol works.

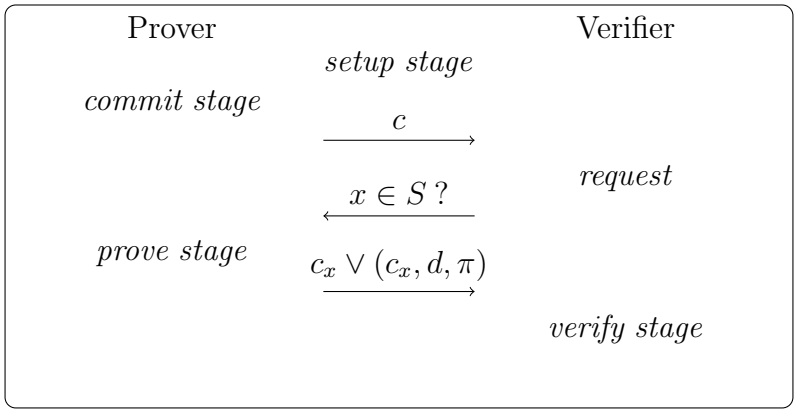


Figure 5.8: Summing the ZKS protocol.

Setup stage. Before starting the protocol, there is a need to define public parameters and public functions. Regarding the functions, there will be two kind of hash functions $\mathcal{H}$, which is an hash function that maps a string of length $k+1$ to a prime number, where k is a security parameter. Thus, $\mathcal{H}_0$ will be $\mathcal{H}_0(x) = \mathcal{H}(x||0)$ and $\mathcal{H}_0$ will be $\mathcal{H}_1(x) = \mathcal{H}(x||1)$. After agreeing on the security parameter k, the Prover and Verifier will take as input and generate the public parameters n, which is an k-bit size Rivest, Shamir and Adleman (RSA) modulus, i.e. a product of two safe primes, and g, a randomly chosen generator.

Commit stage. The Prover must commit the set in order to prevent changes afterwards. Thus, to commit a set $S = \{x_1, x_2, ..., x_m\}$, the Prover randomly chooses a binary string y of k-bit size and computes:

$$u = \mathcal{H}_1(y) \cdot \mathcal{H}_0(x_1) \cdot \mathcal{H}_0(x_2) \cdot ... \cdot \mathcal{H}_0(x_m)$$

$$c = g^u \bmod n$$

The prover will save privately S,u and y, and sends, or publishes, the commit c.

Prove stage. Proving will be different whether if an element x belongs to the set or not. Therefore, if:

- $x \in S$: The Prover will compute $p_x = \mathcal{H}_0(x)$, $u_x = u/p_x$, $c_x = g^{u_x} \bmod n$, and sends c_x to the verifier.
- $x \notin S$: The Prover will compute $p_x = \mathcal{H}_0(x)$. Then, he will choose a pair $(a, b) \in \mathbb{Z} \times \mathbb{Z}$ such that $a \cdot u + b \cdot p_x = 1$, computes $c_x = c^a \bmod n$

and $d = g^{-b} \bmod n$. The Prover will then send c_x, d and a proof π

$$\mathsf{PK}\{(\alpha, \beta) : (c_1 = c^{\alpha} \wedge d = g^{\beta})\}$$

to the verifier.

Verify stage. Since there is two kinds of proofs, there will be also two kind of verifications that the Verifier needs to do. If he receives:

- c_x: this response will imply that $x \in S$. In order to verify, the Verifier will compute $p_x = \mathcal{H}_0(x)$, and then check if

$$c_x^{p_x} = c \,(\bmod\, n)$$

- (c_x, d, π): in this case, the Prover asserts that $x \notin S$. The Verifier will then compute $p_x = \mathcal{H}_0(x)$ and check wheter

$$c_x = d^{p_x} \cdot g \,(\bmod\, n).$$

If it does, he will check if π is a proof that the Prover knows a and b such that $c_x = c^a \bmod n$ and $d = g^{-b} \bmod n$.

5.3.2 How does it fit

Some changes are needed in the workflow presented in Subsection 5.2.2 in order to prevent the creation of fake credentials by RA. In the end, one can see the changes made by comparing Figure 5.9 with 5.5. The first change is when the administrator uploads the list of voters. Instead of sending a list of tokens to RA, Helios will publish a commitment of that list to a Bulletin Board (BB), which may be seen as the commit stage of the ZKS protocol presented above. Afterwards, whenever someone registers with a token t_i, RA will complete the ZKS protocol with Helios over a secure and private channel. Depending on whether the token is valid or not, RA will return to the voter a certificate of registration and the password to access the booth.

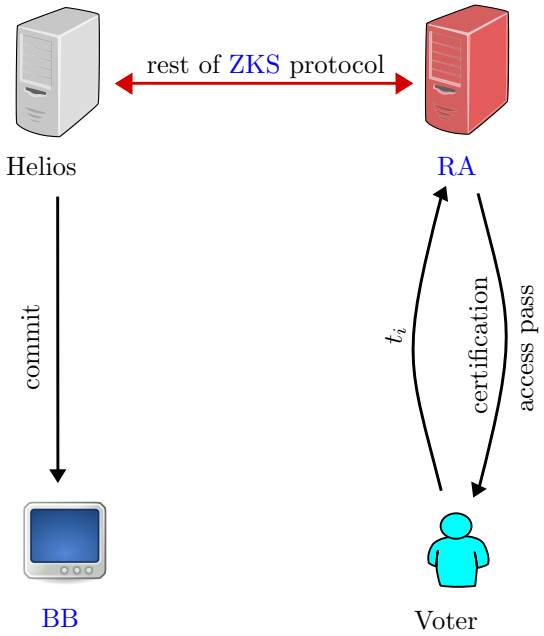


Figure 5.9: Embedding the ZKS protocol.

In essence, we accomplish anonymous registration. This new approach to the registration stage will solve the corruption problem and keeping some advantages of the old scheme:

- RA cannot produce fake credentials, even if he finds the link to the virtual booth.
- Ballot stuffing and vote selling by RA is prevented.
- An adversary is still not able to fool RA, in order to receive fake credentials or the token to access the virtual booth.

5.4 Protecting the booth

Like it was mentioned in the introduction of this thesis, the biggest con argument against Internet Voting (*i*-voting) systems, is that there is no way to provide virtual booths to the voters that are tamper proof. This leads to high level assumptions, which is a reason enough to prevent its use in a high stake elections. For instance, Helios assumes that the voter casts his vote in a trusted platform. When used in the International Association for Cryptologic Research (IACR) contest, there were severe critics to its use, which culminated in a well documented attack in [Estehghari and Desmedt, 2010]. The attack used a well known flaw of a tool used by Helios, which enable a scripting attack

to change the randomness used when computing the ElGamal ciphertexts. Afterwards, the Helios team fixed the flaw, but the attack served as a warning of the amount of problems that such service has to take into account.

Nevertheless, the path towards the creation of a secure *i*-voting system should not stop when facing such problems. New solutions arise everyday to prevent them, some simple, as the use of secured connections, and some complex, as creating static code analyzers. In [Adida et al., 2009], the authors suggest a simple way to help preventing casted ballots on tampered computers. They provide Live CDs for every voter who suspects that his computer might have been compromised with virus. However, there are some kind of attacks that do not depend on a compromised computer. In this section, we give an example of such attack and propose a solution that helps to prevent it.

5.4.1 Phishing attack

Phishing is a kind of a social engineering attack that aims to steal personal information of a victim. Like the term suggests, the attack starts with a lure or deceit to fool the victims. Typically, phishing attacks are targeted to customers of banks and online payment services. In those cases, the attacker, claiming to be a representative or an employee of the bank, sends a fraudulent e-mail with an Uniform Resource Locator (URL) to a forged website, very similar to the real one. When accessing that website, the customer will involuntarily give his access codes to the attacker. Afterwards, the attacker can, at very least, impersonate the customer when accessing the original bank's website. Although this attack is very successful, there are several ways to prevent it. For instance:

- Attention to details - changes on the URL or on the workflow and interactions.
- Augmented authentication systems - for instance, SMS codes of confirmation.
- Use of certificates - for secure and authenticated connections.

However, these standard solutions do not apply in case if the attacker is Helios itself. Imagine that Helios wants to change the randomness that encrypts the ballots. Doing so, he will be able of decrypting them even if there are trustees involved in the election. Thus, when a voter accesses the virtual booth, Helios will send tampered JavaScript (JS) scripts that normally would provide randomness on the encryption. Since the attack will only affect the

encryption scripts, the user will not be able to verify changes on the workflow. Since the attack is not related with authentication mechanisms, its level of security does not affect the attack. And finally, the use of certificates only serves to protect against attacks from third parties. Hence, we propose a solution that may solve this particular problem, but in turn, will affect the workflow of the election, specially how it is prepared.

5.4.2 Code Signing

To better understand the approach to this problem, it helps putting questions from the voters' point of view:

1. *How can I know if the received code has not been tampered?*
2. *How can I know that the code was written from a supposed source?*
3. *How can I know if the code received is equal for any other voter and not a "special" version?*

The answer to the first and second questions is using a technique called code signing [Schiavo, 2010]. This technique uses PKC to guarantee integrity and authenticity of the received code. The complete process involves an external entity usually called Certificate Authority (CA). After Helios[2] produce the necessary code to create the virtual booth, he will sign each one of the needed scripts using a certificate issued by the CA, in order to create a Digital Signature (DS), as seen in Figure 5.10.

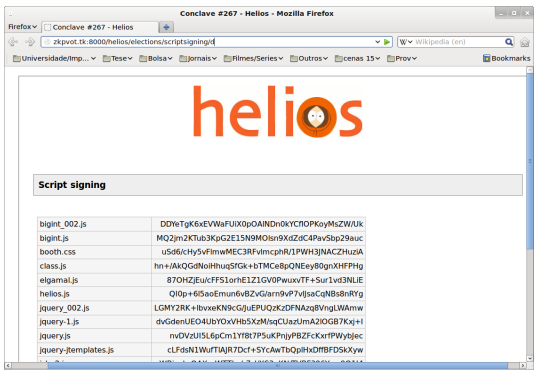


Figure 5.10: Result of signed scritps.

Then, when a voter wants to check the validity of the received code, he will have the means to do it, by verifying if the certificate was issued by a trusted

[2]In this case, Helios is the "author" of the created code.

CA and if the digest of each script matches the correspondent DS. Solving the last of the voters' concerns involves changing the preparation of the election. Now, the DS will be publish in a secure BB by Helios at any time before the election starts. Thus, the voter will know that the version he received is the same as the one signed before the election had started and that any other voter will be verifying the code with the same DS.

5.5 Adding more mobility

For last, recall that to evaluate an *e*-voting systems, just like in the IACR contest from Section 4.1, not only security matters. So, it was decided to improve the *mobility* requirement of Subsection 2.3.2. We developed a prototype Android application to let voters cast their ballots using their smartphones. The first decision to make was choosing the approach:

1. Adapt and optimize the Helios website to the Android's browser.
2. Implement an application to work with the Helios' Application Programming Interface (API).

The choice fell on the second approach due to the following reasons. First, when studying Helios, it was used an Android's browser to cast a vote. From the experience, it took almost a minute to encrypt a ballot with one question and three candidates. A native application will have better performances. The second reason is that although Helios is increasing in modularity, there are still a few dependencies and unnecessary repeated code, which result in a harder maintenance and testing. Recall that Helios is an open-source project with a lot of changes since its first version. Moreover, the fourth version of Helios is probably coming out anytime soon, accordingly to its documentation[3]. Finally, although adapting *views* is easier, we wanted to seize the opportunity to work directly with the cryptographic tools, even if, at the end, the result was far from being efficient and robust. The next subsection gives an overview of the application and some of the interesting details, and the last subsection shows some figures and descriptions of the prototype.

Overview Recall the several JSON structures presented throughout Chapter 4, mainly in Section 4.3. Since our approach was to take advantage of the Helios' API, we modeled our tool to create objects directly related to those structures. All the set resultant from that approach jointly with a `Client` class

[3]Some of the documentation of the future Helios v.4: http://documentation.heliosvoting.org/verification-specs/helios-v4.

formed the `mVoting` package. Two other packages were created, the `CryptoAPI` package, to deal with cryptographic operations, and the `HttpAPI`, responsible for the web communications. Figure 5.11 shows how the application is currently organized.

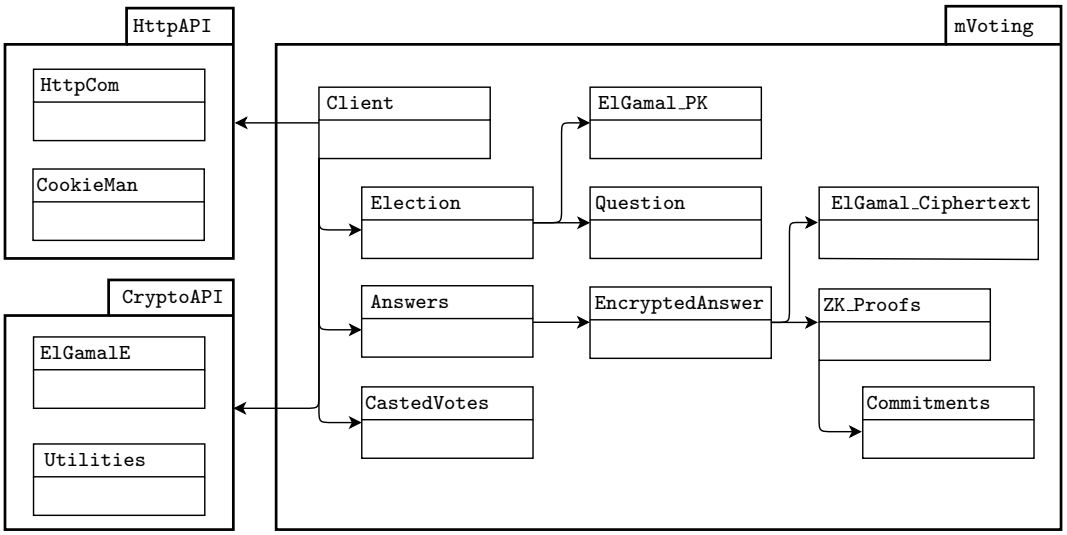


Figure 5.11: Minimal representation of the application's packages.

The following list concerns the most interesting details of the implementation:

- As it was mentioned above, for each JSON structure, we developed a correspondent Java class, as shown partially in Example 5.2, which uses the `Encrypted_Answer` from Example 4.7 and transforms it to the `Encrypted_Answer` class.
- The usage of the Gson[4] library to convert between JSON structures and Java objects. The library provides simple methods like `to_JSON()` and `from_JSON()`, ideal for the development. Simple usage would be `Question e = gs.fromJson(teste, Question.class);`.
- The communication with the Helios server is done by making `GET` and `POST` requests. All the respective methods are in the `HttpCom` class, which mostly makes use of the `Java.net.HttpURLConnection`. The main difficulty, which is not yet perfectly solved, is the management of the sessions. By now, the solution provided is using the `Java.net.CookieManager` on the `CookieMan` class. Unfortunately, due to some technical difficulties when creating `cookies`, we could only submit encrypted ballots with hard coded connections.

[4]Google-Gson: http://code.google.com/p/google-gson/.

- Finally, regarding the `CryptoAPI` package, it is worth mention that includes several utilities on the `Utilities` class, for instance "ready-to-use" hash functions, and the tools necessary to encrypt ballots with the `ElGamalE` class.

```
{
  "choices": [
    {"alpha": "..", "beta": ".."},
    {"alpha": "..", "beta": ".."},
    ...
    {"alpha": "..", "beta": ".."}
  ],
  <Individual_Proofs>
  <Overall_Proof>
}
```

```
package mvoting;
import ...
/** ... */
public class Encrypted_Answer {
    public ArrayList<ElGamal_Ciphertext> choices;
    public ArrayList<ZK_Proof> individual_proofs;
    public ArrayList<ZK_Proof> overallproof;
    ...
}
```

Example 5.2: Conversion of JSON structure to Java class.

Views and activities Finally, in this subsection we show the different views of the virtual booth. Each one was created to be as much similar as possible to the browser's version. Each tag represents the information to be changed when generating an particular election booth.

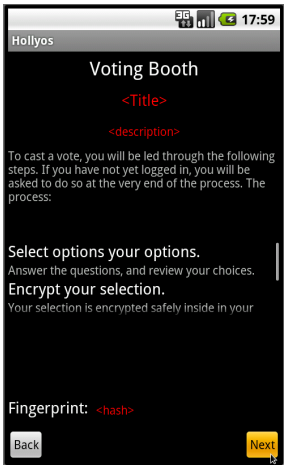


Figure 5.12: Home frame.

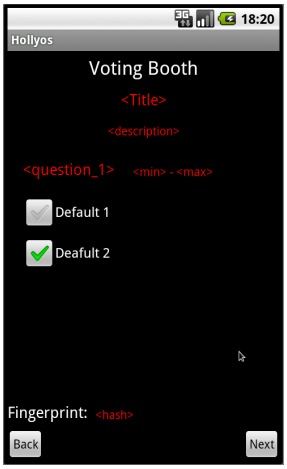


Figure 5.13: Picking options.

Figure 5.14: Encrypting.

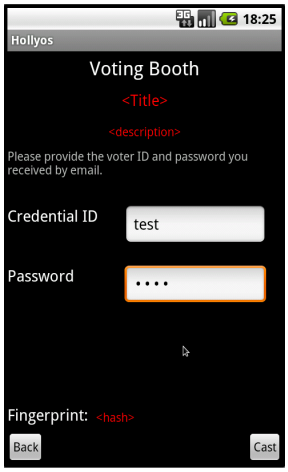


Figure 5.15: Inserting the credentials.

5.6 Summary and final notes

In this chapter we present several proposals to improve the Helios system. It is important to note that this does not detract value from it, since it is difficult to find a system that is better prepared and conceived than Helios. Before summarizing what problems we addressed and the chosen techniques to overcome them, it is interesting to note that there were other things also considered, for instance:

- The trustees problem: recall that in the current version of Helios, if by any reason one of the trustees "loses" his share of the secret, then there is no way to decrypt the final tally. This is an interesting problem to solve, since, as it was mentioned in Subsection 3.4.4, the best scenario of Secrete Sharing (SS) is to have a defined threshold of trustees needed to decrypt a ciphertext. We did not approach this problem since it is already a feature promised for Helios v.4. Apparently, their solution uses Lagrange coefficient.

- Developing an entire API: Helios provides an API for anyone who wishes to verify the correctness of an election. In fact, the solution presented in Section 5.5, has a lot of information hard coded. In other words, it does not offer flexibility and may not even work with small changes on Helios. Our idea is why not provide an API that offers support for anyone that wishes to develop a virtual booth on their own? This would also benefit the problem of Section 5.4, provided that the premiss "a greater supply of booths means a smaller chance of using a tampered one" is valid.

Summing up, we start by fighting coercion. Helios does not provide coercion-resistance measures, so with a double scheme password, we, at very least, extended significantly the amount of time needed to an adversary be able of coercing a voter. Next, we introduce a new entity to the system. Despite of not being a new approach, [Adida et al., 2009] also has an external authority to deal with registrations, our goal was completely different. In our design, we remove the assumption of trusting Helios for privacy, replacing it by a lower assumption of no collusion. This design leaved an open problem which we try to solve with the use of ZKS, a kind of ZKP which goal is to prove that an element belongs to a set without revealing any information other than that.

Finally, to increase security and reliability on Helios regardless the *e*-voting scheme involved, we create a feature that applied code signing to all the scripts that compose the virtual booth. This way, each voter may check the validity of the booth, whether it was sent by a trusted entity or not. For last, an Android application was implemented with sole purpose of adding mobility to Helios voting system.

Chapter 6

Conclusions

> *Abstention means you stayed at home or went to the beach. By casting a blank vote, you're saying you have a political conscience but you don't agree with any of the existing parties.*
>
> *Jose Saramago*

In this chapter, we present a brief summary and the conclusions of our work. We end this thesis by stating what we expect to do in the future with this work's contributions.

Conclusions on the presented work. Our goal for this thesis was to study and improve a remote Electronic Voting (*e*-voting) system. Thus, it was necessary to approach the *e*-voting problem from the beginning. We first presented a conceptual perspective by showing: different kinds of voting; the main stages of an election; and the usual entities involved in elections. Next, we presented the general requirements and the security properties that a voting system should achieve. All the definitions were based on a large set of articles found in literature. Regarding general requirements, the definitions found were adapted for voting systems, but they could easily be generic for almost any kind of electronic system. In turn, the security properties had a significant degree of difficulty in finding proper definitions. The main reason was the lack of standards to define them. Nevertheless, we chose the most concise definitions for each property or requirement found. A general conclusion found in literature states that is impossible to achieve *universal verifiability* and *coercion-resistance* without strong assumptions. Concerning remote *e*-voting, this problem aggravates since most of those assumptions are physical.

Cryptography is essential to design *e*-voting schemes, specially the Internet Voting (*i*-voting) type. From the research made, we identified several cryptographic primitives that belong to most *e*-voting protocols. The next list sums, in a rough way, how the schemes could be classified:

- Using Homomorphic Encryption (HE): Schemes that use this primitive usually have the advantage of not needing to decrypt single ballots. This could also be translated in computational benefits. On the other hand, most of these schemes are limited on the number of available voting types.

- Using Blind Signature (BS): This primitive has the advantage of providing secrecy in a ballot only known by a voter. They are not limited by any voting type. Employing BS usually imply several different entities. Depending of course on the design, this could bring problems regarding collusion and achieving *receipt-freeness*.

- Using Mixed Network (mixnet): Primitives such Zero-Knowledge Proof (ZKP), Secrete Sharing (SS) and Re-Encryption are usually needed to create mixnet protocols. Its original goal was to provide anonymity, and despite its computational limitations, is the most employed primitive in recent schemes.

Among the available systems to study, Helios was the obvious choice. This system has already been chosen to be used in elections of significant importance. Other reasons include: being open-source; fully *i*-voting; and enough documentation. Helios use HE in its design. It offers privacy under the assumption "Trust Helios" and complete integrity of the results. No measures for *coercion-resistance* were provided. There are several ongoing works that aim to improve Helios: improving usability; achieving everlasting privacy; provable ballot privacy; and adding features to provide mixnet. For walking towards a complete secure Helios, it was necessary modify its scheme in order to lower the assumptions and provide mechanisms to protect the voter. From the changes performed on the scheme, we believe to have:

- **Provided anti-coercion measures** - we modify the scheme in order to have two passwords when casting a vote: the *secured-pass* for casting under normal circumstances; and the *coerced-pass*, to cast under situations of coercion. The resultant system turns coercion impractical. Only in extreme cases a coercer is able to illegally cast a vote. Unfortunately, the assumptions needed are too strong. We needed to extend the "Trust Helios" for integrity too and a secure registration stage.

- **Lowered Helios crucial assumption** - we added an extra entity responsible to manage the electors. Since a pseudonym is used instead of a real name when casting a vote, Helios will no longer be able to bind a voter to a vote. Like the previous changes, this also has an assumption, "No collusion between Helios and the extra entity". However, is a weak assumption when comparing to the original one. Some other advantages result in adding the new entity as presented before.

- **Anonymous registration** - The major drawback of the previous solution was concerned the possibility that the new entity became corrupt. This lead to problems such as ballot stuffing or credential selling. Our solution to this problem was using Zero-Knowledge Sets (ZKS). This primitive combined with minor changes in the previous scheme prevent the extra entity of being corrupt without being discovered.

Changing the scheme is not the only way to improve security in Helios. Empowering the system with security techniques, such as the use of certificates or secure connections, is necessary. Without making an analysis regarding this kind of security, we tried to contribute by using code signing, a technique to protect and maintain the authenticity of the source code. At the end of this work, we believe there is still much room to improve Helios, for instance in most of the general requirements. Finally, our final conclusion is that while there is no consensual solution to the *e*-voting problem, or more specific the remote type, a voting system should provide solutions to any kind of security level or voting types.

Future work. A very important conclusion in [Adida et al., 2009], states that:

> *The biggest lesson, of course, is that no matter the voting system, each election is a significant project on its own. One cannot simply install a piece of software and expect an election to run smoothly.*

We agree that there are some elections with peculiarities which makes practically impossible to develop a secure voting system that applies to all kinds of elections. Nevertheless, each system should be designed to handle the maximum possible types. Here, modularity plays an important role, since it is necessary for a system to be adaptive without high costs. We generally see modularity applied in voting systems to provide different types of voting. Helios for instance, provides Referendum and Multiple Candidate Selection. But, as stated in our previous conclusion, we believe that we should extend the modularity to cover different types of security, as the ones proposed in this work.

With this in mind, our next step is to develop a system from the scratch, having Helios as inspiration for what it achieved. In this new system, we also wish to incorporate the idea of an Application Programming Interface (API) to develope different kinds of voting booths.

References

Ben Adida. *Advances in cryptographic voting systems*. PhD thesis, Cambridge, MIT, MA, USA, 2006. AAI0810143. **Cited** on pages 19, 28 and 29.

Ben Adida. Helios: web-based open-audit voting. In *Proceedings of the 17th conference on Security symposium*, SS'08, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association. **Cited** on page 54.

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 *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections*, EVT/WOTE'09, pages 10–10, Berkeley, CA, USA, 2009. USENIX Association. **Cited** on pages 53, 54, 55, 78, 86, 92 and 95.

Aditya, Lee, Boyd, and Dawson. An efficient mixnet-based voting scheme providing receipt-freeness. In *International Conference on Trust and Privacy in Digital Business (TrustBus), LNCS*, volume 1, 2004a. **Cited** on pages 21 and 40.

Riza Aditya, Colin Boyd, Edward Dawson, and Byoungcheon Lee. Implementation issues in secure e-voting schemes. In E Kozan, editor, *Proceedings of Abstracts and Papers (On CD-Rom) of the Fifth Asia-Pacific Industrial Engineering and Management Systems (APIEMS) Conference 2004 and the Seventh Asia-Pacific Division Meeting of the International Foundation of Production Research*, pages 1–14, Gold Coast, Australia, 2004b. Queensland University of Technology. **Cited** on page 15.

Josh Benaloh. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, EVT'06, pages 5–5, Berkeley, CA, USA, 2006. USENIX Association. **Cited** on page 53.

Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, pages 544–553, New York, NY, USA, 1994. ACM. ISBN 0-89791-663-8. doi: 10.1145/195058.195407. **Cited** on pages 3 and 38.

Josh C Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, PODC '86, pages 52–62, New York, NY, USA, 1986. ACM. **Cited** on page 38.

David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. Adapting helios for provable ballot privacy. In *Proceedings of the 16th European conference on Research in computer security*, ESORICS'11, pages 335–354, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-23821-5. **Cited** on page 68.

Dan Boneh and Philippe Golle. Almost entirely correct mixing with applications to voting. In *ACM Conference on Computer and Communications Security*, pages 68–77, 2002. doi: 10.1145/586110.586121. **Cited** on pages 33 and 40.

Philippe Bulens, Damien Giry, and Olivier Pereira. Running mixnet-based elections with helios. In *Proceedings of the 2011 conference on Electronic voting technology/workshop on trustworthy elections*, EVT/WOTE'11, pages 6–6, Berkeley, CA, USA, 2011. USENIX Association. **Cited** on page 68.

Ran Canetti, Cynthia Dwork, Moni Naor, and Rafi Ostrovsky. Deniable encryption. Cryptology ePrint Archive, Report 1996/002, 1996. **Cited** on page 34.

David Chaum. Blind signatures for untraceable payments. In *CRYPTO'82*, pages 199–203, 1982. **Cited** on page 30.

David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 89–105, London, UK, UK, 1993. Springer-Verlag. **Cited** on page 35.

David Chaum, Peter Y. A. Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In *Proceedings of the 10th European conference on Research in Computer Security*, ESORICS'05, pages 118–139, Berlin, Heidelberg, 2005. Springer-Verlag. **Cited** on page 40.

David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity ii: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *Proceedings of the conference on Electronic voting technology*, EVT'08, pages 14:1–14:13, Berkeley, CA, USA, 2008a. USENIX Association. ISBN 888-8-88888-888-8. **Cited** on page 52.

David Chaum, Aleksander Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan T. Sherman, and Poorvi L. Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. *IEEE Security & Privacy*, 6(3):40–46, 2008b. **Cited** on pages 49 and 105.

David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981. ISSN 0001-0782. doi: 10.1145/358549.358563. **Cited** on page 28.

Jacek Cichoń, Miroslaw Kutylowski, and Bogdan Weglorz. Short ballot assumption and threeballot voting protocol. In *Proceedings of the 34th conference on Current trends in theory and practice of computer science*, SOFSEM'08, pages 585–598, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-77565-X, 978-3-540-77565-2. **Cited** on page 49.

Jeremy Clark and Urs Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. *IACR Cryptology ePrint Archive*, 2011:166, 2011. **Cited** on page 73.

Veronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In *Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium*, CSF '11, pages 297–311, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4365-9. doi: 10.1109/CSF.2011.27. **Cited** on page 68.

Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 103–118, Berlin, Heidelberg, 1997. Springer-Verlag. ISBN 3-540-62975-0. **Cited** on page 38.

L.F. Cranor and R.K. Cytron. Sensus: a security-conscious electronic polling system for the internet. In *System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference on*, volume 3, pages 561 –570 vol.3, jan 1997. doi: 10.1109/HICSS.1997.661700. **Cited** on pages 15, 22 and 23.

Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc. **Cited** on page 27.

Saghar Estehghari and Yvo Desmedt. Exploiting the client vulnerabilities in internet e-voting systems: hacking helios 2.0 as an example. In *Proceedings of the 2010 international conference on Electronic voting technology/workshop on trustworthy elections*, EVT/WOTE'10, pages 1–9, Berkeley, CA, USA, 2010. USENIX Association. **Cited** on pages 54 and 85.

Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP J. Inf. Secur.*, 2007:15:1–15:15, January 2007. ISSN 1687-4161. **Cited** on page 26.

Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251, London, UK, 1993. Springer-Verlag. **Cited** on pages 3 and 39.

Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. *Secure Distributed Key Generation for Discrete-Log Based Cryptosystems*. 1999. doi: 10.1007/3-540-48910-X_21. **Cited** on page 36.

Ed Gerck, C. Andrew Neff, Ronald L. Rivest, Aviel D. Rubin, and Moti Yung. The business of electronic voting. In *Proceedings of the 5th International Conference on Financial Cryptography*, FC '01, pages 243–268, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44079-8. **Cited** on pages 3, 4, 19 and 23.

Oded Goldreich. Zero-Knowledge twenty years after its invention. *Electronic Colloquium on Computational Complexity*, 2002. **Cited** on page 34.

Rüdiger Grimm, Robert Krimmer, Nils Meißner, Kai Reinhard, Melanie Volkamer, and Marcel Weinand. Security requirements for non-political internet voting. In Robert Krimmer, editor, *Electronic Voting*, volume 86 of *LNI*, pages 203–212. GI, 2006. ISBN 978-3-88579-180-5. **Cited** on page 19.

Douglas W. Jones. Early requirements for mechanical voting systems. In *Proceedings of the 2009 First International Workshop on Requirements Engineering for e-Voting Systems*, RE-VOTE '09, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-4100-6. doi: 10.1109/RE-VOTE.2009.3. **Cited** on pages 19 and 103.

Juang and Lei. A secure and practical electronic voting scheme for real world environments. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, 1997. **Cited** on page 21.

Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, WPES '05, pages 61–70, New York, NY, USA, 2005. ACM. ISBN 1-59593-228-3. **Cited** on page 48.

Fatih Karayumak, Maina M. Olembo, Michaela Kauer, and Melanie Volkamer. Usability analysis of helios - an open source verifiable remote electronic voting system. In *Proceedings of the Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE)*, page 5, Berkeley, CA, USA, 2011. USENIX Association. **Cited** on page 67.

M. Fatih Karayumak. Usability analysis and interface improvement of the end to end verifiable remote-electronic voting system helios. Diploma thesis, Technische Universitat Darmstadt, 2010. **Cited** on page 67.

Aggelos Kiayias, Michael Korman, and David Walluck. An internet voting system supporting user privacy. In *ACSAC*, pages 165–174, 2006. **Cited** on page 49.

K. Kim, J. Kim, B. Lee, and G. Ahn. Experimental design of worldwide Internet voting system using PKI. 2001. **Cited** on page 39.

Miroslaw Kutylowski and Filip Zagorski. Scratch, click and vote: E2e voting over the internet. Cryptology ePrint Archive, Report 2008/314, 2008. **Cited** on page 49.

Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. *IACR Cryptology ePrint Archive*, pages 116–116, 2012. **Cited** on page 54.

Triinu Mägi. *Practical Security Analysis of E-Voting Systems*. PhD thesis, Tallinn University of Technology, 2007. **Cited** on page 15.

Rebecca T. Mercuri. *Electronic vote tabulation checks and balances*. PhD thesis, Philadelphia, PA, USA, 2001. AAI3003665. **Cited** on pages 15 and 22.

Silvio Micali, Michael Rabin, and Joe Kilian. Zero-knowledge sets. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS, pages 80–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2040-5. **Cited** on page 82.

Andrew C. Myers, Michael Clarkson, and Stephen Chong. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368. IEEE, May 2008. **Cited** on page 48.

Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka, and Tatsuaki Okamoto. An improvement on a practical secret voting scheme. In *Proceedings of the Second International Workshop on Information Security*, ISW '99, pages 225–234, London, UK, UK, 1999. Springer-Verlag. **Cited** on page 39.

Akira Otsuka and Hideki Imai. Unconditionally secure electronic voting. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 107–123. Springer, 2010. ISBN 978-3-642-12979-7. **Cited** on page 22.

Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, UK, 1992. Springer-Verlag. ISBN 3-540-55188-3. **Cited** on page 37.

Stefan Popoveniuc and Benjamin Hosp. An introduction to punchscan. In *Towards Trustworthy Elections*, pages 242–259, 2010. **Cited** on page 52.

Jean-jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Thomas A. Berson. *How to Explain Zero-Knowledge Protocols to Your Children*. 1989. doi: 10.1007/0-387-34805-0_60. **Cited** on page 108.

Zuzana Rjaskova. Electronic voting schemes, 2002. **Cited** on pages 21 and 40.

Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à voter: a voter-verifiable voting system. *Trans. Info. For. Sec.*, 4(4):662–673, December 2009. ISSN 1556-6013. doi: 10.1109/TIFS.2009.2033233. **Cited** on pages 15, 20, 21, 49, 50 and 105.

Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth. In *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'95, pages 393–403, Berlin, Heidelberg, 1995. Springer-Verlag. ISBN 3-540-59409-4. **Cited** on pages 22 and 53.

Krishna Sampigethaya and Radha Poovendran. A framework and taxonomy for comparison of electronic voting schemes. *Computers & Security*, 25(2):137–153, 2006. **Cited** on pages 15, 20 and 22.

Jay Schiavo. Feature: Code signing for end-user peace of mind. *Netw. Secur.*, 2010(7):11–13, July 2010. ISSN 1353-4858. doi: 10.1016/S1353-4858(10)70093-3. **Cited** on page 87.

Bruce Schneier. *Applied cryptography - protocols, algorithms, and source code in C (2. ed.)*. 1996. **Cited** on pages 3 and 36.

Michael Shamos. Electronic Voting - Evaluating the Threat, 1993. **Cited** on page 19.

Ben Smyth. Replay attacks that violate ballot secrecy in helios. *IACR Cryptology ePrint Archive*, pages 185–185, 2012. **Cited** on page 68.

Rui Xue, Ning-Hui Li, and Jiang-Tao Li. Algebraic construction for zero-knowledge sets. *J. Comput. Sci. Technol.*, 23(2):166–175, March 2008. ISSN 1000-9000. doi: 10.1007/s11390-008-9119-x. **Cited** on page 82.

Appendix A

Complements

A.1 Brief History of Voting Technologies

Throughout this section, we introduce some of those mechanisms, in order to show the evolution of voting systems. It is important no note that the use of one of those mechanisms do not preclude the use of another on the same elections. In fact, it is very common to see them working together to overcome some disadvantages that one of them may have. The use of several different mechanisms on a same election are usually called a "hybrid system".

A.1.1 Secret Paper Ballot

Commonly known in literature as the *Australian Ballot* [Jones, 2009], due to its origins in Australia, around 1850, the secret ballot was the first voting method of the "modern world". Apart from being the oldest, the general scheme of the secret paper ballot is still the most worldwide used.

Using an election as an example, in this method, the voter starts by authenticating himself with an official authority. If he is verified as an able voter, i.e. he is registered and still has not voted before, he is given a paper ballot and conducted to a private cabin to vote. This way, the voter is free to secretly make his choice, not suffering intimidation or bribery from third parties. His choice is made by filling a paper ballot, usually writing the name of the candidate or picking a check-box. It is important to mention that the ballots must be specifically designed to

Figure A.1: Ballot box.

eliminate any bias and also to prevent anyone from connecting voter to his ballot. Made his choice, he proceeds to place his vote, without revealing any of its info, in a sealed box.

A.1.2 Lever machine

Lever machines were the first kind of direct-recording voting systems. These mechanical machines were based on late 1800s patents by Thomas Edison and others. As the name suggests, these kind of systems do not work with any kind of receipts, i.e. the lever machines left no record of an individual voter's intent. Thus, when one votes, his vote is directly recorded on the counting mechanism and there is no need of reading or interpreting the vote.

After the process of authentication (in this case it works in a similar way as the secret paper ballot), the voter is conducted to a lever machine. When he enters the machine, he pulls a lever, closing the door of the machine and unlocking the voting levers, guaranteeing the voter's privacy. There is a set of levers, each one of them corresponds to a candidate. The voter can make his choice, freely to change his mind if the machine has already a pulled lever. Then, if the machine has only one pulled lever, he is able to pull an external lever. This lever will process the vote, rotating a wheel that works as a counter, open the door and lock the levers, which in turn return to its initial positions[1].

A.1.3 Punchcards

Lever machines had the huge drawback of being large and expensive machines. Consequently, inventors start looking for cheaper and more usable solutions. Punchcards were the next choice as voting technologies around 1950s [2]. These kind of tools employ a card (or cards) and a small device for creating votes. Voters punch holes in the cards opposite their candidate or ballot issue choice. This was only the mechanism for filling a ballot. Casting it would be like paper ballots, where a voter would place his card in a sealed ballot box.

There are two kinds of punchcards: the "votomatic" card and the "datavote" card, as seen in Figure A.3. Punchcards had the advantage of automatic and quick tallying of votes. Nevertheless, there were some problems regarding its use, such as verifying the vote or malfunctioning machines that let half holes in the card.

[1] See http://inventors.about.com/library/weekly/aa111300b.htm.

[2] A complete history of Punchcard:http://whatis.techtarget.com/tutorial/history-of-the-punch-card.html

Figure A.2: Lever Machine.

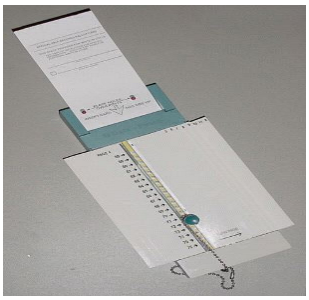


Figure A.3: Datapunch.

A.1.4 Optical Scan

Optical scan machines[3] were also invented around 1950s. In this technology, voters usually fill a circle or complete an arrow to mark their choices in a ballot. Afterwards, they cast the ballot by inserting it in a machine. The machine itself will count the votes and store it in several memory cards. Later, the ballots will be available for checking the validity of the results.

The sophistication of these machines have evolved in such way that are usually considered as the best tool for conducting elections[4]. In fact, two of the most prominent voting systems use this technology in their *e*-voting schemes: Prêt à Voter [Ryan et al., 2009] and Scantegrity [Chaum et al., 2008b].

A.1.5 Direct-recording Electronic

The last significant contribution to voting technologies was the Direct-recording Electronic (DRE), which is kind of the electronic version of the Lever machine. DRE machines were the first to employ and explore advanced cryptographic techniques, regarding voting problems, specially when combined with networked systems, where anonymity between communications are required. Their use has been studied extensively, hoping to achieve the ultimate secure system. Nevertheless, these devices are becoming massively used, replacing the older technologies.

The DRE machines is user-friendly, providing several advantages when compared with the other technologies. The voter is faced with a ballot displayed in the screen and makes his choice simply by pressing buttons or even a

[3]For more information: http://en.wikipedia.org/wiki/Optical_scan_voting_system

[4]Technology news: http://www.eweek.com/c/a/Government-IT/Optical-Scanners-Winning-War-of-Voting-Machines/

touchscreen, which when pressed, it triggers processes that records the vote's information. After the election is over, it calculates a tabulation of the voting data stored in a removable memory component and a printed copy.

Figure A.4: Optical Scan.

Figure A.5: DRE Machine.

A.2 Additional Lessons on Cryptography

With this section, it is intent to provide additional information about the cryptographic primitives described in Chapter 3.

A.2.1 Hard Problems

Hard computational problems are the base for every secure protocol created concerning Public-key Cryptography (PKC). According to the European Network of Excellence in Cryptology[5], these problems may be categorized into six different groups:

1. Discrete Logarithms
2. Factoring
3. Product Groups
4. Pairings
5. Lattices

[5]More information about hard problems: http://www.ecrypt.eu.org/wiki/index.php/Main_Page

6. Miscellaneous

The following descriptions are for specific hard problems related to previous techniques described in Chapter 3.

Discrete Logarithm Problem (DLP). Given $h \in \mathcal{G}$, we want to compute x such that $h = g^x$.

This problem belongs to the first group. Its use can be found in several protocols, for instance Schnorr signatures, DSA signatures and the ElGamal Encryption (ElGamal) encryption, as shown in 3.1.1. The best known algorithm for DLP in general is the parallel Pollard rho method, which has complexity $O(\sqrt{r})$.

RSA Problem. Given a positive integer n which is the product of at least two primes, an integer e coprime with $\psi(n)$ and an integer c, we want to find and integer m such that $m^e = c \,(\mathsf{mod}\, n)$.

This problem belongs to the second group. Its use can be found in the example of the RSA signature in Section 3.3. Although there is some discordance in literature, some researchers argue that this problem can be reduced to another problem called *Factoring Problem*. If so, the best known algorithm to solve it is the Number Field Sieve, which has complexity $L_N(1/3, c)$ for some constant c.

A.2.2 Paillier Encryption

The Paillier cryptosystem is another case of an encryption scheme with homomorphic properties, more specifically, the additive homomorphism, just like Exponential ElGamal Encryption (EEG). It was invented by Pascal Paillier and is a kind of PKC. Its security is based on the hardness of computing n^{th} residue classes. Figure A.6 sums up how the scheme works.

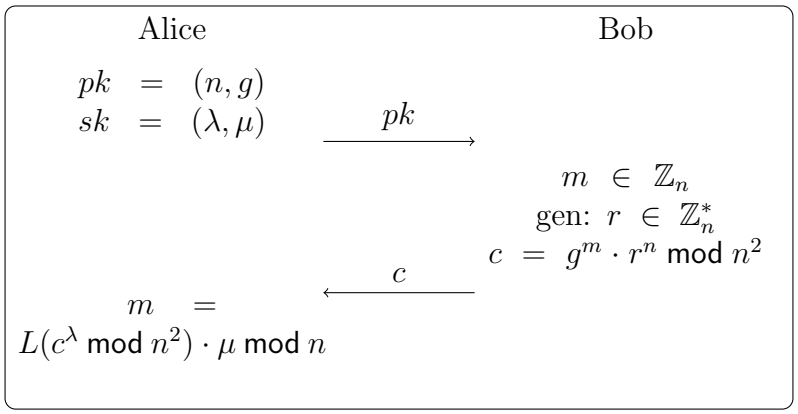


Figure A.6: Paillier Encryption.

To generate the public key pk and secret key sk, Alice must choose two large primes p and q such that the great common divisor between $p \cdot q$ and $(p-1) \cdot (q-1)$ is equal to one. Next, she computes $n = p \cdot q$ and λ, which is the least common multiple for q and p. Then, she generates $g \in \mathbb{Z}_{n^2}^*$. Finally, she calculates $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$ where $L(u) = (u-1)/n$. The pair (n, g) will be the pk and the pair (λ, μ) will be the sk.

A.2.3 Informally Explaining ZKP

In literature, there is a story invented by Jean-Jacques Quisquater to help non-cryptographers understand the ideas of ZKP [Quisquater et al., 1989]. It starts with two young children, Pedro (the *prover*) and Vânia (the *verifier*), finding a cave (Figure A.7). After exploring for a while, they both found that the cave is circular, but they were unable to go around it because of a secret locked red door.

Two weeks later, Pedro, being a smart little fellow, discovers the code to open the red door. Vânia, envious of Pedro's newfound knowledge , asks for the code and says that she is even willing to trade something with him for that information. After closing the deal, Vânia, who is always suspicious of Pedro, says that she will not do her part of the deal until she is certain that Pedro knows the code. Pedro swears that he will tell her the

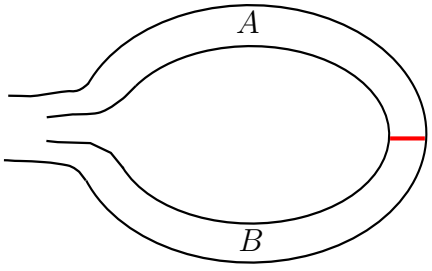


Figure A.7: The cave.

code, but never before she fulfills her
part of the deal, because it is not the first time that he comes out empty-handed. So they both design a scheme in which Pedro can prove that he knows the code without revealing it.

First, they label the paths that leads to the red door as A and B. Vânia will wait outside until Pedro enters the cave and goes, following randomly A or B path, near the red door. Afterwards, Vânia enters the cave and screams, randomly, the path that she wants Pedro to return from the red door. Pedro, who is very honest, opens the door, if necessary, and returns from the chosen path.

However, Vânia still does not trust Pedro. After all, there was a 50% chance that Pedro had chosen to follow the right path since the begging. So they repeat the process over and over again, and Pedro always came out from the right path. After 20 repetitions, the probability of Pedro being a liar was near 0% ($(1/2)^{20} = 0.000000954$).

Finally, Vânia trusts Pedro enough to know that he really knows the code. In the end, she receives the code, but Pedro never got his share of the deal because of his "bad attitude" problems. He should never have called her a "stubborn little lady"... Twenty times.

A.2.4 Blind Schnorr Signature

Another transformation of regular Digital Signature (DS) to BS can be seen in the so called Blind Schnorr Signature.

Let G be a subgroup of Z_n^* of order q, for some value n and some prime p, and a generator $g \in G$. These are the parameters agreed by a *Signer* and a user in a Schnorr signature scheme. They also agree on a public *hash function* H, whose domain is $\{0,1\}^*$ and range is Z_q, and the *Signer* generates a number $x \neq 0$, which will be his secret key, and calculates $y = g^x$, which in turn will be his public key.

The scheme starts with the user requesting a message m signed by the *Signer*. Initially, the *Signer* computes a random $k \in Z_q^*$, calculates and sends the commitment $r = g^k \pmod p$. He proceeds by calculating e, which is the result of H over the concatenation of m and r. Finally, the *Signer* computes

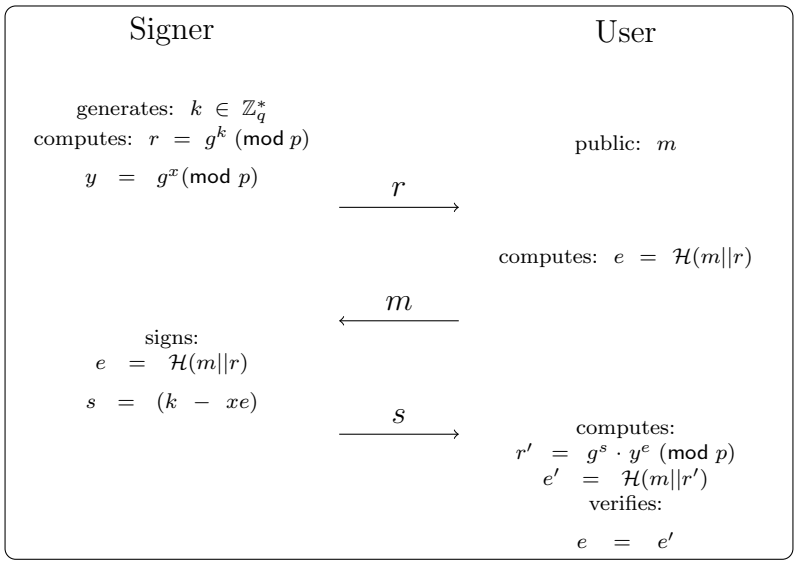


Figure A.8: Classical Schnorr Signature

$s = (k - xe)$ and sends the signature (s, e) to the user. The user calculates $r' = g^s \cdot y^e \pmod p$. The concatenation of m and r' will be the argument for the *hash function* H, which results in e'. The final comparison $e = e'$ will prove the authentication of the signature s sent to the user.

The blind Schnorr signature follows the same steps as the regular Schnorr signature until the *Signer* sends the commitment r. After receiving it, the user generates two random elements $\alpha, \beta \in Z_q$ and computes an $r' = r \cdot g^{-\alpha} \cdot y^{-\beta} \pmod p$. Afterwards, he calculates an e', which is the *hash function* H over the concatenation of m and r' modulo q, and $e = e' + \beta$, and sends e to the *Signer*. Finally, the *Signer* returns s such that $g^s \cdot y^e = r \pmod p$. The user can later verify the authenticity of the returned value by calculating $s' = s - a \pmod q$ and checking if e' is equals to returned value of H over the concatenation of m and $g^{s'} \cdot y^{e'} \pmod p$.

A.2.5 Guillou-Quisquater identification scheme

The Guillou-Quisquater is an example of the many uses of ZKP. It is an identification scheme based on the Rivest, Shamir and Adleman (RSA) problem, as seen in subsection3.3. Figure A.10 demonstrates how the protocol works.

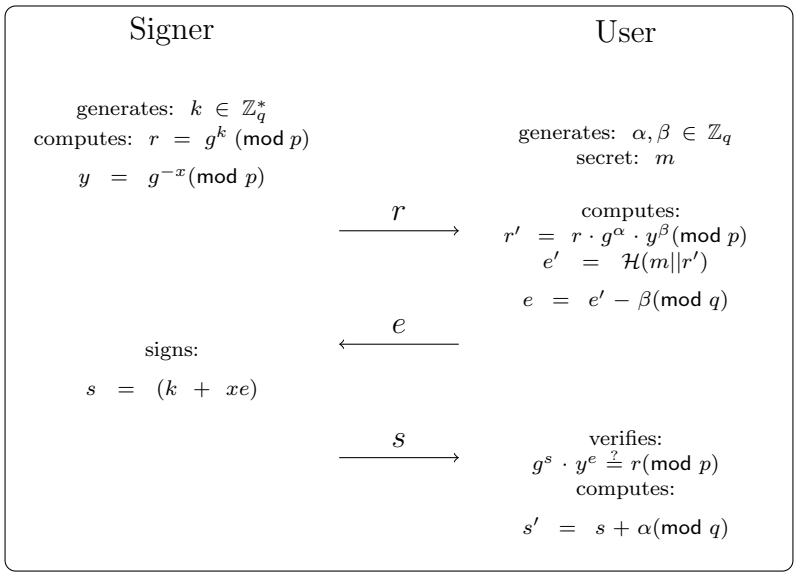


Figure A.9: Blind Schnorr Signature

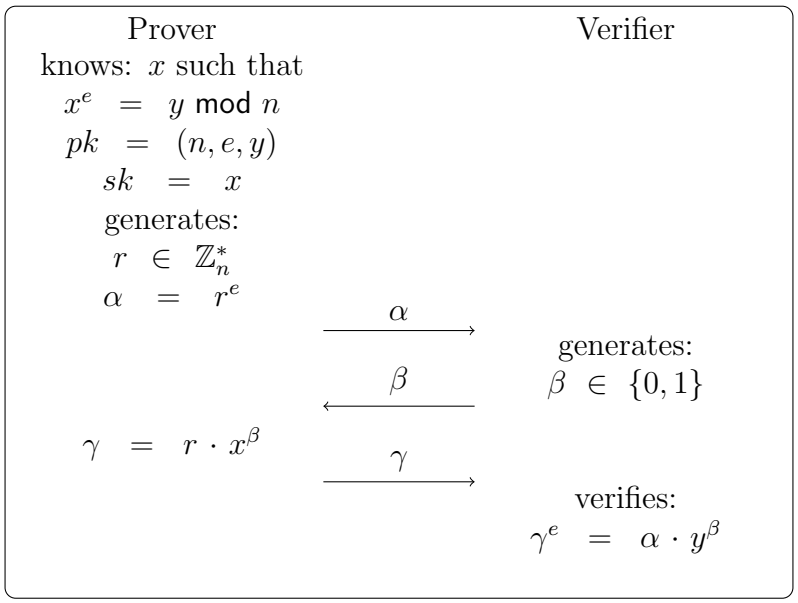


Figure A.10: Guillou-Quisquater identification scheme.

This process can be repeated n times until de verifier is satisfied with the odds. An interesting use for this protocol is in problems related to anonymous access control. The verification step can be proven as follows:

$$\gamma^e = \alpha \cdot y^\beta$$

$$
\begin{aligned}
(r \cdot x^{\beta})^{e} &= \alpha \cdot y^{\beta} \\
r^{e} \cdot x^{e\beta} &= r^{e} \cdot y^{\beta} \\
x^{e\beta} &= y^{\beta} \\
x^{e\beta} &= x^{e\beta}
\end{aligned}
$$

A.3 Less Important Figures

This section contains the figures with less importance in this thesis, but with interesting information worth to show. It will be divided according to the different themes approached throughout the dissertation.

A.3.1 Helios

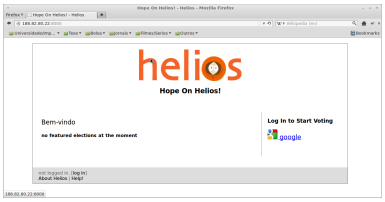


Figure A.11: Login page.

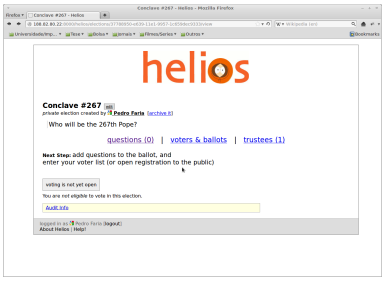


Figure A.12: Home page of the election.

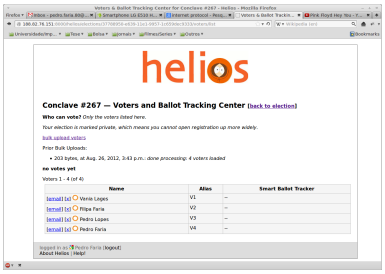


Figure A.13: After uploading the list of voters.

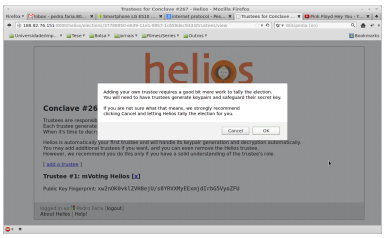


Figure A.14: Adding trustees warning.

A.3.2 Mobile Figures

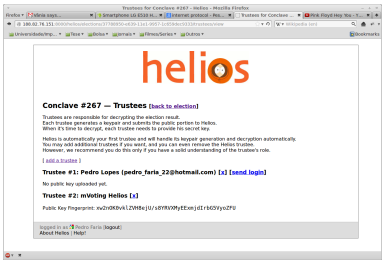


Figure A.15: Waiting for the upload of the public key.

Figure A.16: Invitation to participate in the election as a trustee.

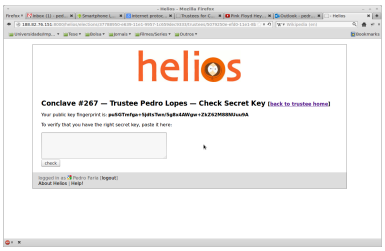


Figure A.17: Checking the secret key.

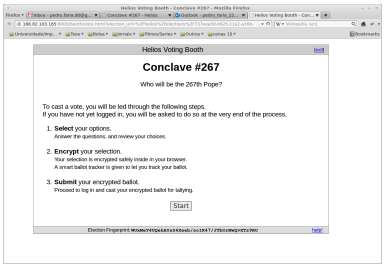


Figure A.18: Voting instructions.

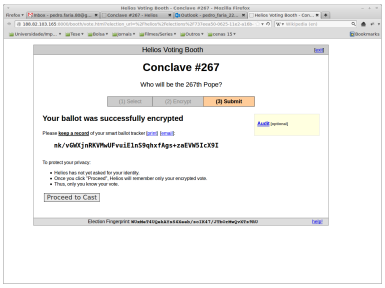


Figure A.19: Encrypted vote fingerprint.

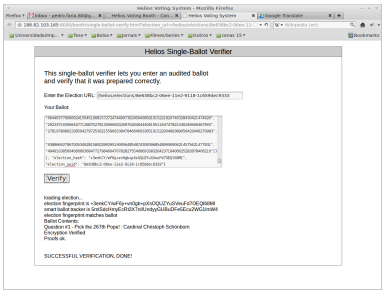


Figure A.20: Auditing a ballot.

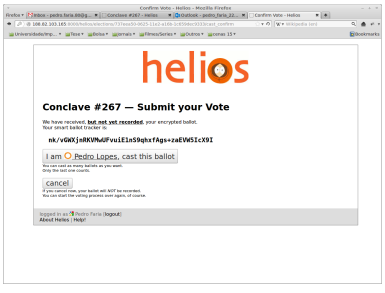


Figure A.21: Confirmation of credentials.

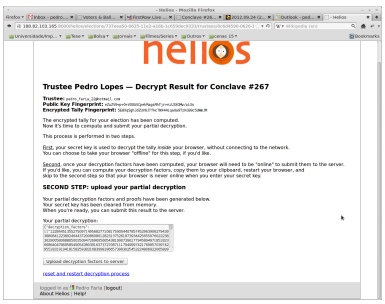


Figure A.22: Upload of the secret key.

Figure A.23: Final verification.

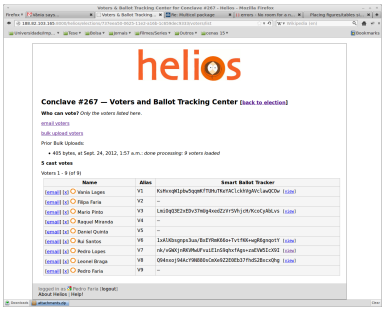


Figure A.24: Smart ballot tracker with the administrator's point of view.

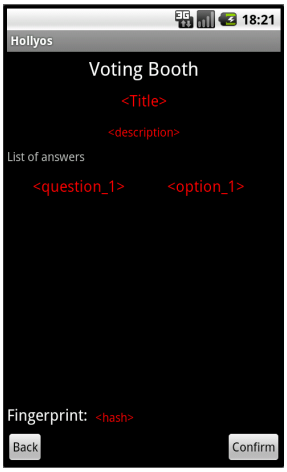


Figure A.25: Default booth home.

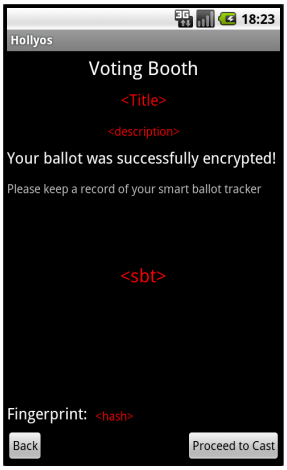


Figure A.26: Successful warning.