



E-Voting on Fenix

Fernando Mário Machado Marques

Thesis to obtain the Master of Science Degree in
Information Systems and Computer Engineering

Supervisor(s): Prof. Ana Almeida Matos
Prof. Jan Cederquist

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. Ana Gualdina Almeida Matos
Member of the Committee: Prof. Miguel Nuno Dias Alves Pupo Correia

November 2017

Dedicated to both of my grandmothers.

Acknowledgments

I would like to thank my family and friends for all the support given to me when I was feeling less motivated.

I would like also to thank my supervisors Ana Almeida Matos and Jan Cederquist who were always available to provide help and to guide me when I needed.

A special thanks to Professor Carlos Ribeiro, Professor Luis Guerra e Silva and Professor Paulo Mateus for the meetings and insight they gave me on various topics needed for the creation of this document.

I would also like to thank Instituto Superior Técnico which allowed me to make this research.

And finally, I would like to thank the reader, who is taking its time and reading this document.

Resumo

Esta tese considera o problema de implementar um sistema de votação eletrónica para ser utilizado no Instituto Superior Técnico, integrando este com o sistema informática já presente, Fenix. Como tal, este trabalho assume que resistência à coerção não é de preocupação principal. É requisito do sistema que este também se integre com o protocolo de votação em papel já existente de forma a que utilizadores com menos experiência ou que não confiem no sistema eletrónico também tenham a possibilidade de exercer o direito de voto. Esta tese apresenta um protocolo de votação eletrónica híbrido baseado num protocolo já existente, Belenios, integrando este com o sistema de autenticação do Fenix e com o sistema de votação em papel também já presente. De seguida são apresentados alguns detalhes da implementação do protocolo e uma análise teórica e prática que foram efetuados de forma a provar a correta execução e segurança do protocolo. No fim, uma conclusão sobre a possibilidade da utilização de tal protocolo é apresentado seguido de algum trabalho futuro que pode ser realizado de forma a melhorar o protocolo.

Palavras-chave: votação electrónica, híbrido, Belenios, protocolo

Abstract

This thesis considers the problem of implementing an e-voting system on Instituto Superior Técnico, integrating this to the already existing university information system Fenix. As such, the work presented works under the assumption that coercion resistance is not of primary concern. It is required that the system be able to integrate the already present paper voting protocol in order to provide to those who have less experience or less trust in the e-voting system a means in which they may exercise their right to vote. This thesis presents a hybrid e-voting protocol based on the already existing Belenios protocol, which is then integrated with the Fenix authentication system and with the already existing paper voting protocol. It then present some details on the implementation of said system and a theoretical and practical analysis which were performed in order to provide proof of correctness and security of the protocol. In the end, a conclusion is presented regarding the possibility of using such a protocol and some future work which will improve the protocol is presented.

Keywords: e-voting, hybrid, Belenios, protocol

Contents

Acknowledgments	v
Resumo	vii
Abstract	ix
List of Tables	xv
List of Figures	xvii
1 Introduction	1
1.1 Problem	1
1.2 Objectives	2
1.3 Requirements	2
1.4 Contributions	3
1.5 Thesis Outline	4
2 Context and related work	5
2.1 E-Voting overview	5
2.1.1 Public key encryption	6
2.1.2 Digital signatures	7
2.1.3 Secure Channels	7
2.2 Existing Protocols	8
2.2.1 Prêt à Voter	8
2.2.2 Scantegrity II	8
2.2.3 Remoteegrity	9
2.2.4 VeryVote	9
2.2.5 Civitas	10
2.2.6 EVIV	11
2.2.7 TrustVote	12
2.2.8 sElect	13
2.2.9 Helios	15
2.2.10 Helios-C	17
2.3 Discussion of suitability	18
2.4 Chapter Summary	20

3	Protocols	21
3.1	Baseline Protocol A: Current Paper Voting protocol	21
3.2	Baseline Protocol B: Belenios	22
3.3	H-Belenios	24
3.3.1	Set up Phase	25
3.3.2	Voting Phase	25
3.3.3	Tally Phase	26
3.3.4	Audit Phase	27
3.4	H-Belenios Correctness	27
3.5	Chapter Summary	30
4	Implementation	31
4.1	Architecture	31
4.2	Adopted Technologies	32
4.3	Back-end Details	33
4.3.1	Database	33
4.3.2	Connecting to Fenix	37
4.3.3	Connecting to the mail Server	38
4.3.4	Administrator Functions	39
4.4	Cryptographic Details	40
4.4.1	Cryptographic Parameters Generation	40
4.4.2	Random Number Generation on the Front-End	40
4.4.3	Zero Knowledge Proofs	41
4.5	Front-End Details	41
4.5.1	Server Administrator Pages	42
4.5.2	Trustee Pages	44
4.5.3	Voter Pages	44
4.6	Specification Fulfilment	46
4.6.1	Set Up Phase	46
4.6.2	Voting Phase	46
4.6.3	Tally Phase	47
4.6.4	Audit Phase	48
4.7	Chapter Summary	49
5	Usability evaluation	51
5.1	Tests Description	51
5.2	Results	52
5.3	Discussion	54
5.4	Chapter Summary	55

6	Conclusions and Future work	57
	Bibliography	59
A	Ballot Example	A.1

List of Tables

2.1	Summary of Advantages/Disadvantages of different protocols	19
3.1	Set up phase specification	25
3.2	Voting phase specification for electronic voting	26
3.3	Voting phase specification for paper voting	26
3.4	Tally phase specification	27
5.1	Simulated paper votes for question "Which of these movies would you like to see?"	52
5.2	Simulated Paper Votes for question "Which of these books do you like best?"	52
5.3	Recorded Electronic Votes for question "Which of these movies would you like to see?" .	53
5.4	Recorded Electronic Votes for question "Which of these books do you like best?"	53
5.5	Tally for question "Which of these movies would you like to see?"	53
5.6	Tally for question "Which of these books do you like best?"	54

List of Figures

4.1	Architecture used	32
4.2	Models used	34
4.3	Fenix OAuth2 Authorization Configuration	38
4.4	Homepage of the e-voting application	42
4.5	Create Election Page	42
4.6	Election Administrator Control Panel	43
4.7	Smart Bulletin Board before publishing the results	45
4.8	Smart Bulletin Board after result publication	45
4.9	Example voter list after result publication	45

Chapter 1

Introduction

Voting has been one of the main ways for decision making in the current times and has been for some time now. It is used by many to make decisions being these small and sometimes insignificant to very important ones which can impact the way society works, for example government election or referendums [1]. For a long time, society has used the paper voting system, in which ballots are sheets of paper which is used to cast a vote by putting them in a ballot box and then someone counts them, being the result publish later. This system has worked for ages and although it works, it also has some disadvantages, for example how can a voter check that their vote was actually count correctly?

Electronic voting, also denoted as e-voting, came in order to so try and solve some problems which existed with paper voting and although it has some advantages, it also has some disadvantages. The main advantages that can be seen from e-voting is the possibility of voting remotely (thus people who have low mobility or even those who could go the the voting booth due to some restriction are able to cast a vote), it reduces the time and effort of managing an election also cutting on staff cost, but most of all some protocols allow the voter to verify that their vote is being counted correctly. Although this may only seem to bring advantages, e-voting also brought some disadvantages. One of those is that vote coercion is easier in e-voting, another is that a protocol which is not implemented correctly may bring about security flaws that could impact the result of an election.

In the end, it is not correct to say if e-voting or the classical paper voting is better since both has their advantages and disadvantages but the truth is that e-voting has gain popularity over the year and better protocols have been made in order to tackle the disadvantages it has.

1.1 Problem

E-voting has been used in many situations already, from governmental elections [1], the municipal elections [2] and even some university elections [3, 4].

As such it would be advantageous to bring an e-voting system to Instituto Superior Técnico since currently elections are done using the classical paper format (like the university presidential elections). This would allow for easier management between different campuses and would cut time costs from

the election. It would also bring some integrity advantages which are currently not supported with the existing protocol. Unfortunately, as seen, not every user is comfortable with changing from the classical paper voting to an electronic one. As such this thesis tackles the problem of implementing an e-voting protocol on Instituto Superior Técnico and to use the existing resources available, such as the integrated campus management system to the advantage of the e-voting protocol.

1.2 Objectives

The main objective of this thesis is the specification and implementation of a prototype of a hybrid e-voting protocol, which is based on the already existing paper voting protocol used at Instituto Superior Técnico and an existing e-voting protocol. The decision of making a hybrid protocol mitigates the problem mentioned in the previous sections. The protocol will also connect with the already present authentication system provided by Fenix.

The protocol will be specified in this thesis and an analysis on the assumptions and security properties of the same will be available. The prototype of the implementation will be available for consultation and the results of user testing will also be made clear.

1.3 Requirements

A solution such as this will have the three main types of requirements:

Security Requirements

These are the requirements which have to do with the security of the system. Being a voting system, these are of primary concern. The main requirements which should be kept in mind are the following:

1. The votes must remain anonymous in the system, meaning that it must not be able to associate a vote with the person who voted.
2. The system must provide integrity of the ballots which are saved before the tally.
3. The system should ensure a high availability and reliability.
4. The system should be end-to-end verifiable.
5. No person should have the full power to decrypt the election data.
6. Only eligible voters should be able to vote.
7. The system shall count all the votes and do so correctly.
8. No voter should be able to vote more than once.
9. All steps of the election will be available to audit by either the voters or the election administrator.

Functionality Requirements

These are the requirements which affect how the system will function and feel to the user. The requirements are:

1. The voter will have to authenticate himself using Fenix.
2. Voting may be configured in order for it to be only available during certain time intervals.
3. The user interface should be easy to use.
4. Every voter shall be given the choice between paper voting and electronic voting.

Architectural requirements

These requirements are those that affect the architecture of the system. These can be due to hardware limitation, documentation limitation or other. The architecture requirements which were considered relevant are:

1. The user must be able to vote in a computer with an internet connection, preferably a safe computer.
2. No other devices should be necessary for the voter to vote.
3. The system should be portable in order to simplify integration and adaptation to other organizations.
4. The voting system should be run on a limited amount of servers, with the maximum number being two (which is the number of servers available).
5. The design and implementation shall be documented and available to the public.

1.4 Contributions

As such, in this thesis, a specification and implementation of hybrid e-voting protocol called H-Belenios, which uses as baseline protocols the current paper voting protocol used at Instituto Superior Técnico [5] and the Belenios e-voting protocol [6] (chosen due to its simplicity and worked according to the requirements), is presented.

As such, we can summarize the main technical contributions as:

- A new hybrid voting system that integrates a modified version of the Belenios e-voting protocol (in which voting credentials are generated on demand) and a classical paper voting protocol.
- Accurate enunciation of the preservation of properties of the baseline protocols and their proofs.
- Proof of concept of applicability to an academic setting, by the design of an architecture for implementing the proposed solution that fits the implementation requirements of an existing organization.

- Implementation of a working prototype of the system.
- An usability evaluation in order to prove that users would adopt and use such a system.

A paper explaining the protocol specification was published in INForum 2017 Atas do Nono Simpósio de Informática [7]. The code of the implementation and a wiki which will be developed as the implementation evolves will be available in the github project page [8].

1.5 Thesis Outline

This thesis starts by presenting a bit of context on the theme of e-voting, the problem it tackles and its objectives.

It then presents Chapter 2 where a more in depth explanation of e-voting is presented and some concepts which are fundamental in order to understand the protocols. This chapter then presents a list of existing protocols, explaining how these work and end with a section describing the main advantages and disadvantages of each of these protocols according to the requirements mentioned in Section 1.2.

The next chapter presents the solution to the problem at hand. It starts by explaining the two protocols that served as baseline for our solution. Afterwards it presents a specification of the solution followed by a chapter where correctness of the protocol is proven based on the assumptions and security properties of the two baseline protocols.

Chapter 4 presents the details of the implementation. In these are the technologies that were used, how the databases are organized, how the connection to the Fenix authentication system is done and other. During this chapter, the most relevant cryptographic details are presented, followed by a description of the front-end done for the system. In the end, it presents a close relation between the implementation and the specification and the architecture that was used during development and testing.

Following this chapter, the results of an usability tests are presented. The tests are described and a list of results is presented. It is also present a discussion on the results were a reflection on what can still be improved is done.

In the end, a conclusion to the thesis is available. In this, a reflection on the overall work is done and a list of possible future work is available.

Chapter 2

Context and related work

In this chapter we present the background research done for this thesis. We start by explaining how e-voting can be divided and the greatest difficulties in implementing an e-voting protocol. Following we present some concepts which are commonly found in e-voting protocol and finally we present some of the protocols that were researched in order to produce our solution. At the end of this section we present an overview of the protocol and the main advantages and disadvantages these have in accordance to our objectives described in section 1.2.

2.1 E-Voting overview

Before explaining in more detail some existing systems that have been implemented for e-voting, it is first necessary to understand some concepts and ideas used by these.

Most of the e-voting systems which are currently used have the following common definitions:

Voter is the entity which is going to interact with the system in order to cast the vote. This entity is the main user of the system and can be either honest or dishonest. The idea of honesty of the voter comes from whether he will follow the established rules of the election or not.

Candidate is an entity to which the voter can vote on.

Adversary is an entity whose main objective is to disrupt the correct execution of the election. They may try to access votes, coerce voters or even cast unauthorized votes in order to change the end result of the election.

Ballot corresponds to the sealed collection of votes that a voter has cast. These are necessary to be kept secret so that no one can identify who voted for whom.

Bulletin Board is a component on which information regarding the election is published. This can be the tally, voters who have cast a ballot, the list of all eligible voters and other relevant information. The information published here differs from system to system due to the different security requirements for each of these.

Since the act of voting normally deals with sensitive information, it is to no surprise some of the most important requirements for these system are security requirements, more specifically confidentiality and integrity. Confidentiality deals with the problem of keeping the votes and sensitive information of the system secret while integrity deals with the problem of keeping the information correct and tamper free. As such these two security requirements make it difficult to design a e-voting protocol since maintaining a good balance between confidentiality and integrity is difficult. If we see the extreme cases, we can cast a vote without encrypting it and thus it is easy to see if the election was tempered with but we don't have confidentiality, or we keep every bit of data secret and it is impossible to check if something was tempered with [9]. As such a good balance between these two is necessary. In order to maintain these requirements, system used various techniques like mix-networks or homomorphic encryption based on public key encryption schemes (see section 2.1.1), digital signatures (see section 2.1.2), secure channels (see section 2.1.3) and others.

When talking about e-voting system, it is also important to understand the idea of verifiability. We have two main types of verifiability in voting systems, individual verifiability and universal verifiability. These are differentiated based on who can verify, if only a specific user can verify the data then we have individual verifiability, otherwise if everyone can verify the data then we have universal verifiability. System can have individual verifiability in some parts, for example in only allowing the election administrator to checking proofs of decryption, and universal verifiability in other parts, for example everyone can check the bulletin board and see the only eligible voters have cast a vote.

There are also the idea of cast as intended, which gives the voter the ability to verify that the ballot was correctly formed before casting it, and count as cast, in which the voter has the ability to verify that the vote which was counted was the same as the one cast. When these two properties are present in an e-voting system, it is said that it is End-to-End verifiable.

2.1.1 Public key encryption

Public key encryption is a cryptographic scheme that makes use of a pair of keys to encrypt data. One of the keys is made public (public key) and can used for encrypting data and the other key(private key) is kept secret(only known to its owner) and is used to decrypt the data received. Some election systems normally use this type of cryptographic scheme in order to encrypt the ballot of the voters. One example of a public key encryption scheme used in e-voting system is the El Gamal public key encryption [10].

Even so two problems arise: what if only one person has the private key and as such he could decrypt any ballot he wanted and how to calculate the tally and maintain privacy.

For the first problem, one possible solution is to have distributed key generation. In this, we have a group of people who generate key pairs and publish their public key. These public keys are aggregated in a general public key used for the encryption purpose. To decrypt the encrypted message it is then necessary the participation of everyone who published their public key, since only they know the private keys. This of course presents another problem. What if someone publishes a public key and then decides to not participate in the decryption process? To solve this Adi Shamir presented a process in

which it is possible to share a secret in a way that only a subset of people is needed to reveal the secret [11]. Pederson went a step further by presenting a way to do so without a third party being involved [12].

The second problem presented earlier is normally solved in two different ways. Either through the use of mix networks or through the use of homomorphic encryption.

Mix networks were first mentioned by Chaum [13] and are used in order to anonymize data by hiding the link between the source and destination of the data. The data is encrypted and passed through a number of servers which decrypt each part of the message. Since the message is encrypted in an onion-like fashion a node on the mix network only knows the previous node from where the message came from and the next node of the mix-network. This way, it is only possible to discover where the message came from if all servers of the mix network are compromised.

Another solution to the privacy problem is to use homomorphic encryption. This type of encryption allows computations to be made on the ciphertext resulting in an encrypted result that when decrypted matches the results of operations performed on the plaintext. For example, in exponential El Gamal, if we have two ciphertexts c_1 and c_2 which correspond to the plaintext p_1 and p_2 then when we multiply $c_1 * c_2$ and decrypt the result, it gives us $p_1 + p_2$.

2.1.2 Digital signatures

Digital signatures are a way of giving authentication, non-repudiation and integrity to a message that was sent. This is important in e-voting systems in order to prove that the person who is sending the ballot is actually an eligible voter and not an adversary or even a dishonest server trying to do ballot stuffing. This makes use of keys which are denoted as credentials. Each participant has a pair of keys, a public and a private one, and the receiver of the message also has the public key.

Digital signatures work by signing a message with the private key of the sender. Then the signature is sent with the message to the receiver. On its end the receiver then verifies the signature with the public key and that it corresponds to the signature of the message that was sent.

There is also the concept of blind signatures introduced by David Chaum. These are used when we want an entity to sign a message without providing the message itself. As such a blinding factor is used in order to blind the message before sending it to be signed. The blinded message is then signed by the entity and sent back to the original sender. The sender can then remove the blinding factor from the signature received and have a correct signature over the message he originally sent.

2.1.3 Secure Channels

Secure channels are important to have in a protocol where there occurs message exchange with sensitive information. These channels try to provide message confidentiality, integrity, authentication and freshness during the communication. To provide these features they may use different techniques like public key encryption, digital signatures, the inclusion of timestamps on messages and other.

One of the most common ways of implementing secure channels is to use TLS or SSL [14, 15] for the communication between two parties. These provide the properties mentioned above, although not

all algorithms used in these are secure, allowing some attacks to be done, making the configuration of these protocol very important.

2.2 Existing Protocols

2.2.1 Prêt à Voter

Prêt à Voter [16] implements verifiability by the use of special ballots. These ballots are divided in half where the left half has the candidates list in random order (different for each ballot) and the right half is the place where the vote is cast with an “X”. This right side also has encrypted information called “onion” which enables the system to reconstruct the order of the candidate list. This information is encrypted in a way that no single party can decrypt this information. The voter then casts the vote and keeps the “onion” as a receipt. When the results are published, the voter can then check if the vote and onion appear on the bulletin board correctly. If not the receipt can be used to challenge the election. The voter can also choose to audit the ballots before casting the vote. This is done by removing the left side of the ballot and asking the system to give the candidate list based on the “onion”. The system also publishes enough information so that anyone can verify every stage of the election.

Some systems implement code voting [17] which corresponds to the use of special codes for the voter to verify that their code was valid. These codes can be associated to candidates or can be confirmation codes that are previously distributed on code sheets.

2.2.2 Scantegrity II

Scantegrity II [18] makes use of ballots with codes which can only be seen using a special ink from a decoder pen. This system is to be used in a polling booth and as such makes assumptions that the ballots are not accessible from the outside. Each ballot is composed of a main body with a list of candidates, which have hidden confirmation codes next to each other, a machine readable serial number and two chits that can be detached from the vote. The left chit has a hidden serial number for the ballot and a space where the voter can write the confirmation code while the right chit only has a hidden serial number. Both of these serial numbers are different from each other. Each confirmation code is cryptographically linked to the candidate for each ballot and this is secure with a key that is shared among trustees. As such it is necessary for a number of trustees to collude in order to decrypt the link. When the voter goes to vote he uses the decoder pen and marks the oval space next to the candidate of choice. Doing so turns the oval space dark revealing the confirmation code for a few minutes until it also turns dark. The voter can spoil the ballot by giving it to a poll worker who without seeing the rest of the ballot removes the right chit and destroys the rest of the ballot. The right chit is then used to count the total of ballots that should be equal to those that were spoiled, audited and tallied. If the voter wishes to cast the ballot, then he gives it to a worker who scans the main body and reveals the serial of the chits. The voter can then leave with the chit. The voter can view on a website if the confirmation code that appears linked to the serial number of his ballot is the same as the one recorded.

For auditing a ballot one of the chits is given at random to a worker and another of the chits is kept with the voter. The ballot is then fully marked and cast allowing for the voter to check in the website that all the confirmation codes there are correct.

Finally anyone can verify the data and tally with the use of software acquired from a trusted source or written by the voter (information for writing the software is provided). In case of those organizing the election they need to appoint an independent auditor to verify the system.

2.2.3 Remotegrity

Remotegrity [19] is an extension to the Scantegrity II system in order to use it remotely. This uses ballots similar to the Scantegrity II ballots but with the codes visible. The ballot is sent by mail with another card that contains one authentication serial, four hidden authentication codes, one acknowledgement code and one hidden lock code (codes are hidden behind a scratch-off surface). The voter inputs the ballot and authorization serials and the system checks if these are not duplicates. The voter then inputs the vote code from the Scantegrity card and one of the four hidden authorization codes. The system checks the code and if it is correct and was not previously used then it sends back the acknowledgement code. The voter checks if the acknowledgement code is correct and casts the vote using the lock code. Since the codes are hidden and are randomly assigned there is a low probability of guessing the correct chosen code. In case the client is corrupt and tried to change the vote code then it would have to guess the code from where the system would not reply the acknowledgement code if it were incorrect. As such the client would only have one try to guess the vote code before the voter could verify that the client was corrupt. In this case the voter would not input the lock code. In case of the system being corrupt then the system would have to use an unused authentication code. As such that can be verified by the voter if he presents the card with the codes still sealed.

2.2.4 VeryVote

VeryVote [20] is a code voting system which uses an adapted version of MarkPledge cryptographic receipts techniques [21] to achieve end-to-end verifiability. This technique has the objective of proving that a particular encryption is the correct encryption of the vote for that candidate. It uses an encoding that is a sequence of numbers where the number 1 corresponds to the selected candidate and 0 for the rest of the candidates. Then for each bit it is calculated a sequence of Ballot Mark Pairs (BMP) which are two El Gamal encryptions of a 1 or a 0. If the bit being encrypted is a 1 then both encryptions of the BMP must be of the same number else both encryptions of the BMP need to be of a different number. To verify this encryption the verifier starts by committing a random challenge. The prover then sends the bit encoding and a bit string (ChosenString) where each bit of the string corresponds to the bit encrypted in both elements of the BMP. The verifier sends to the prover the challenge which must be equal to the one it committed previously. For each bit in the challenge the prover reveals the randomness used to encrypt each BMP on the same index where bit 0 corresponds to the left BMP and bit 1 corresponds to the right BMP. This allows to decrypt the bits of the BMP and arrange them to a string which can be

compared by the verifier to the ChosenString given in the beginning by the prover. As such it proves that the encryption is well done.

The candidate choice is done using the codes which are present in a code sheet which is sent by a third party, for example mail. This code sheet has a list of candidates, a code for each candidate and a confirmation code. The voter will then choose a candidate and insert the code that is linked to the candidate and will also commit a hash. The Voting Machine will then encrypt the vote and display a ChosenString. The voter inserts the challenge and the voting machine computes a proof and outputs a receipt. Finally the voter only has to verify that the confirmation code appears next to the candidate he chose in the receipt.

As for other cryptographic protocols used, the election key are shared between a number of trustees using a threshold encryption scheme with a variation of the El Gamal encryption described by Cramer et al. [22]. For minimization it uses a verifiable re-encryption mix networks like the one described by Neff [21].

One of the main disadvantages in this protocol is that since it is the Election Server that makes the code sheets and since it knows who has cast the vote then it can make associate a code sheet to a voter on the election day.

2.2.5 Civitas

Civitas [9] is coercion resistance e-voting system. In this system there are five main agents:

- the voters;
- the supervisor that administers the election;
- the registrar that authorizes voters;
- the registrar tellers that generate the credentials used by the voters;
- the tabulation tellers that tally the votes.

Each of these agents use a log system with a publicly available write only storage.

Civitas resists coercion with the use of credentials. These credentials are divided in public credentials which the registrar tellers publish on the bulletin board and the private credentials which are given to the voter by the tellers when he authenticates himself. Each teller only has a share of the private credentials being necessary for all the tellers to work together in order to give the full private credential. When voting the voter has to input the private credential. In case of coercion, the voter may choose to input a fake credential. These fake credentials can be generated by the voters using a faking algorithm that can be run locally.

The vote is then submitted with the private credential and a proof that the vote is well formed to some or all ballot boxes. The system has the option of allowing revoting. If this option is enabled then the new vote must have a proof of the credential and the choice of the previous vote that is to be replaced. In case the option is disabled then the duplicates votes are discarded. Civitas also allows for multiple ballot

formats, although write-in votes are not coercion resistant. The vote then passes through a tabulation phase where:

- the proofs are verified;
- duplicates are eliminated;
- the votes are anonymized by the use of a mix network;
- the unauthorized votes are eliminated (votes with invalid credentials);
- the tally is publicly computed.

All tabulation tellers posts proofs of the computations made allowing this protocol to be verifiable. Each tabulation teller also check the proofs as tabulation occurs and stops if a proof does not pass.

2.2.6 EVIV

EVIV [23] is an end-to-end verifiable internet voting protocol. The main feature of this system is that it allows the voter to vote from any computer while at the same time guaranteeing the voters privacy. This is achieved by the use of an external device (smartcard, an USB device or even a secure element in a smartphone) which is called the voters security token (VST) which contains an unique cryptographic key pair and that is used to encrypt the voters ballot. For verifiability it is used the combination between code voting and the cryptographic voter's verifiable vote encryption technique which is used in MarkPledge 3 [24].

The protocol is divided in four phases:

- The voter enrolment phase;
- The registration phase;
- The vote casting phase;
- The public verification and tally phase.

The protocol starts with the voter enrolment phase, which is an offline enrolment of the voter. During this phase the voter identifies himself to the Electoral Commission and receives the VST with his private key and a signature by the Electoral Commission. At the end of this phase all the certificates from the voters are published on the bulletin board.

Sometimes before the starting the election begins the registration phase. During this time the election parameters are set and the voter registers a ballot. It starts by the Electoral Commission publishing the candidate list and the election parameters. An El Gamal shared threshold key is also computed and distributed by the trustees being the data from this computation verified by the Electoral Commission. The voter can also decide to register for the election during this phase. For this the voter must connect the VST to a computer who will then connect to the Election Registrar using a Secure Socket Layer/- Transport Layer Security (SSL/TLS) connection. The VST receives the candidate list and the election

public key. The VST then creates the ballot which is comprised of candidate encryptions in random order and vote Validity proofs which are created using the MarkPledge vote encryption function. It then signs the ballot and send it to the Election Registrar which verifies the ballot and if it is successful then it publishes it on the Bulletin Board. The VST then creates a code card which contains random codes (one for each candidate), and a confirmation code which can be printed or noted down. Finally all ballots published are verified by the Electoral Commission and a signature over the list of ballots is created.

The vote casting can be performed on any computer without the risk of the computer changing the voters vote. Even so the voter must protect his own privacy for example, keep the code card secret.

This phase starts by the issuing a random election challenge which is used for the system to be end-to-end verifiable. This is done by the with a distributed random number generator used by the trustees which is then validated by the Election Commission. The election commission then generates a hash of the number, the electoral roll and the ballot list which is the election challenge.

At this point the voters can start voting by connecting to the Ballot Box using a SSL/TLS connection done by the client application. The voter receives the challenge and the list of candidates and inputs the vote code for the chosen candidate. The choice and the election challenge is sent to the VST who checks if the vote code is valid, and computes the vote and the ballot challenge from the election challenge. The VST also computes a receipt where each candidate has a verification code and the chosen candidate has the verification code equal to that of the confirmation code on the voters code card. The proofs of the creation of the verification codes are concatenated to create a receipt validity. After the confirmation of the voter that the receipt is correct, the signed vote (which has the ballot challenge), the verification receipt and the receipt validity is sent to the Ballot Box who verifies the vote and the receipt validity. If everything is correct then it sends it to the Bulletin Board where all receipt validity data is verified and all votes and receipts are validated by a signature over the list of all the voter-ballot-vote-receipt associations.

In the last phase Independent Organizations start by verifying the election public data and commit that verification by signing the list of all the voter-ballot-vote-receipt associations. The voter can also verify their vote by the use of a verification service from an Independent Organization who will give a copy of the receipt to the voter. The tally is finally computed on the votes that were not protested by the use of a homomorphic aggregation of the votes by the Electoral Commission. Each Independent Organization can also verify the vote aggregation and tally decryption proofs.

2.2.7 TrustVote

TrustVote [25] is a hybrid e-voting protocol with a paper based and e-voting based component. It assumes that the paper voting component is available and allows to revoke previous cast e-votes by casting a paper vote. For the e-voting component they use a improved version of the original FOO92 [26] with some extensions of successor to this protocol.

This protocol has the following entities:

- Voter;

- Administrator;
- Registration authority which checks the eligibility of a voter to cast a vote;
- Key Collectors who have the secret shares of the keys;
- Registration board which lists the eligible voters;
- Voting Board which publishes the votes which were cast;
- Tallier which counts the votes at the end of the election.

The protocol is divided into 5 phases.

In the first phase the administrator sends to the voters and to the tallier an event number and a empty ballot. The number is also sent to both the public boards and the registration authorities. It is also the administrator responsibility to send the ids of the voters to the registration authorities.

After this comes the voter preparation phase where the voter can encrypt his vote and create a voter marker which corresponds to the result of passing the following $id||number||m$ trough a one way function. The id is the voter id, $number$ is the one he received from the administrator and m is a voter secret. This vote marker is used as an anonymous identifier.

The registration phase follows where the voter must get the registration authorities to sign his ballot. This is done using blind signatures and a hash which is calculated with $hash(votermarker||number||votes)$. The registration board is used in order to broadcast the data to be signed in order to avoid the voter sending different data to different registration authorities. This phase ends when the voter has at least the minimum number of signatures delimited by a threshold.

The next phase, denoted as Vote Casting, the voter uses a secret sharing algorithm to share his decryption key with the key collectors and send the ballot to the voting board, where a hash and the signatures are verified. In the voting board, the encrypted vote, the signatures and the key collectors who have the key shares are published.

After the e-vote casting period ends, the paper vote casting period can commence. At a pool booth the voter has to prove his identity. The pool worker checks the registration board to see if the voter has cast an e-vote. If not the voter can vote as usual. If he has cast a e-vote, then the voter needs to give his voter secret, which is then used to compute his voter mark, in order to find his electronic vote and revoke it.

Finally comes the counting phase. In order to calculate the electronic tally, the key collectors send the key shares for each voter to the voting board. The tallier then assembles the key for the votes which are valid, decrypts the ballots and compares the content to an empty see if its valid. The final results are calculated with the sum of the result all the electronic votes with the results paper votes and subtracting the results of the revoked votes.

2.2.8 sElect

sElect [27] is a lightweight verifiable remote voting system. This system has two main ways of verifying votes, human verification and automated verification.

The system is composed of the bulletin board, the voters, the voters supporting device which is normally the browser and platform where this is running on, the authentication server, a number of mix servers and the voting authority.

There are also some assumptions made on the system. One of these is that the channels between the voter supporting devices and the authentication server are authenticated. As such we can assume that only authorized voters can cast ballots. The other assumption made is that for each voter supporting device there exists one authenticated channel and one anonymous channel to the bulletin board.

The protocol has the following phases which we are going to describe in more detail:

- Setup phase
- Voting phase
- Mixing phase
- Verification phase

During the setup phase the election parameters are posted on the bulletin board by the voting authority. Every server generates their keys which are then used for signing messages. The mix servers also generate another pair of keys to use in their encryption method. The public keys are then posted on the bulletin board.

During the voting phase is when the voters cast their votes. During this phase there are also generated the verification codes used in the human verifiability which are discussed further ahead. This verification code is a concatenation of a nonce given by the user (it is important the nonces chosen by different user have a small probability of being the same) and a nonce generated by the voter supporting device. After the verification code is formed the voter supporting device encrypts the vote with the verification code and sends it to the authentication server and if this ballot is valid then the server sends an acknowledgement. In the case of revoting the authentication server sends the old acknowledgement. If the authentication server does not accept the first vote, the voter supporting device tries again and if it does not succeed then it files a complaint. After this phase two lists are published on the bulletin board. One with the valid ballots and another with the identifiers of all voters who have cast a valid ballot.

Next is the mixing phase. Like a normal mix network, each mix server will process the ciphertext list which contains the ballots. During this it will check that the input it received is in lexicographic order, has the correct format, is signed and that it does not contain duplicates. If one of these conditions does not occur it sends a complaint to the bulletin board and stops. This step is called input validation. The mix servers also have a processing step where they decrypt the input with their private key, remove the duplicates and order the result. In case the decryption produces an unexpected result or cannot be decrypted then that entry is discarded. The result is then signed and posted on the bulletin board.

Finally is the verification phase. It is on this phase that the two types of verification occur. In human verification or voter verification, the voter simply checks that the voter nonce appears next to the choice. In case the voter did not vote, then they can check the list in order to make sure that their identifier is not there. In the automated verification or the voter supporting device verification, it is the voter supporting

device that verifies the vote and it is triggered when the voter checks the result of the election. This type of verification depends on the honesty of the device. The voter supporting device first checks if the original submitted plaintext is present in the results published. If not the device checks if the ballot is present in the list delivered to the authentication server. If it is not there then the voter supporting device publishes the acknowledgement received by the authorization server thus proving that the culprit is the server. In case the ballot is present in the authentication server then the device checks if the ballot (now encrypted) is present on the result list of the first mix server. If it is then it then checks the second mix server and does the same to the remaining mix servers (uses receipts in order to identify the ciphertext.). In case the ballot is not on one of the lists then the voter supporting device anonymously (in order not to tell how the voter voted) sends a message to the bulletin board which demonstrated that the mix server in question did not function correctly. It is said that the voter accepts the election if the voter nor the voter supporting device complains about the results.

2.2.9 Helios

Helios [28] is an open audit web based voting system meaning that anyone, even a person who cannot vote for that election can audit the election and verify that everything is running accordingly. This voting system is intended for use in low coercion elections and as such it does not offer much coercion resistance.

The first version of Helios is based on the Benaloh vote-casting approach. Privacy is guaranteed only if you trust the server, but on the other hand integrity is guaranteed even if the server is corrupt since anyone can verify the election. Helios is composed of three main components:

- A Ballot Preparation System
- A Bulletin Board
- A Sako-Kilian mix-net [29] based on the El Gamal re-encryption.

The ballot preparation system is the component that allows for the encryption of the votes. It records the voters choices in the election and encrypts these, committing the ballot by displaying the hash of the ciphertext created. In case the voter wishes to audit that ballot the ballot preparation system displays the ciphertext and the randomness used during the encryption. The ballot preparation system then ask the voter to generate a new encryption for the choices. When the voter chooses to seal the choices the ballot preparation system discards the plaintext and the randomness and asks for the voter to authenticate himself. In case the authentication is successful then the encrypted vote is recorded as the voters vote.

On the Bulletin board of the Helios System, the votes are displayed next to a voter identifier be it an username or an identification number. All of the processing data is also available to everyone on the bulletin board in order for them to download and verify the election.

For anonymization Helios uses a Sako-Kilian mix-net based on the El Gamal re-encryption. Each server of the mix-net receives a number of inputs and re-encrypts them. Each server also creates a

“shadow mix” which is used to prove that the mix encrypted the input correctly. A verifier can then ask the shadow mix for the encryption factors and permutations which a dishonest mix server only has 50% chance of guessing correctly. In order to increase integrity, each mix server has more than one shadow mixes thus decreasing the probability of guessing. Since the protocol above is complex it is transformed using the Fiat-Shamir heuristic[30] in which the challenge bits are computed as the hash of all shadow mixes. The decryption of the El Gamal encryption is using the Chaum-Pedersen protocol [31]. The result of these proofs is then posted on the Bulletin Board.

The overall process of the protocol can be described as the following:

The voter can create as many ballots as he wants and verify these. When he is satisfied with the audits, the vote is encrypted with a new randomness and he authenticates himself which casts his vote. The bulletin board posts the encrypted vote next to the voters name to the public. As the election ends, the votes are shuffled and the shuffling process can be verified. After some time the votes are decrypted and a tally is calculated. The results and all the data is finally published on the bulletin board.

Still some security problems arise with this implementation of Helios:

- Helios can cast votes for users by changing the ciphertext of a voter or authenticating himself as one of the voter since he knows the usernames and password of the voters.
- Helios may also show the voter a corrupt ballot thus leading the voter to thinking that he is voting for a candidate when in reality he is voting for another person.
- A dishonest voter may replay the vote another voter.

Some of these problems are mitigated with the use of the auditing tools available to the public thus decreasing the probability of these types of error not being detected.

In version 2 of Helios [32] the protocol has some differences. In this version of the protocol there is more than one trustee which generate a public key pair (based on the additive property and distributed encryption of El Gamal [22, 33]) and an election officer who publishes the election parameters on the bulletin board. These parameters are the public part of the trustees' key, the proof of correct construction, the candidate list, the list of authorized voters and the election fingerprint which is a hash of all the previous parameters. The protocol then runs as follow:

1. The voter gets the election parameters and calculates the fingerprint which he can check with the one available at the bulletin board
2. The voter votes creating a ballot encrypted with the trustees' public key and proof that the vote is permitted. This ballot is displayed to the voter.
3. The voter can audit the vote like in the previous version.
4. When the vote is to be cast, it is sent to the election office who authenticates the voter, checks if the voter is authorized to vote and checks the proof and publishes the vote like in the previous version.
5. Anyone can verify the ballots in the bulletin board are permitted votes.

6. After some time the election officer combines the ballots still encrypted and publishes the tally which is also encrypted. This tally can also be verified.
7. Each trustee decrypts part of the tally and publishes proof of correct decryption.
8. Finally the election officer decrypts the tally and publishes the result.

The third version of Helios is only as extension of the previous where there were added some practical features. Still this implementation has a major flaw. The information displayed on the bulletin board corresponds to the voter id, the votes ciphertext and two signature of knowledges. The problem is that an adversary might submit a ballot with the same ciphertext and signatures as another voter. Since the signatures are valid the system will count this vote as legitimate. If we take the simplified case in which there are only three voters and two of them are honest, the third voter could choose one of the previous voters vote and recast it as his own and thus he would know who that voter voted for breaking the vote privacy property. To avoid easy detection the adversary might even change the ballot without changing the vote by exploiting the malleability of ballots.

Some of the solutions proposed for this problem are:

- Ciphertexts and signatures of knowledge have unique representations.
- Ballots should not contain the same ciphertext, and those that contain the same ciphertext should be rejected.
- Add the identity of the voter could be used in the creation of the signature of knowledge.
- Votes could be bound to the private key of the voter.

2.2.10 Helios-C

An implementation to solve this problem was made with the name of Helios-C [34] which stands for Helios with Credentials. It is composed of five main components:

- The election administrator who publishes the eligible voters, the list of candidates and the result of the tally.
- The registrar who creates and distributes the credentials of the voters.
- The trustees who tally and publish the final result.
- The voters.
- The bulletin box where information is published and processed.

This implementation uses an El Gamal encryption in order to utilize the homomorphic properties present in it, uses the Schnorr signature scheme [35] and non interactive zero knowledge proofs to prove the correctness of the algorithms. It also consists of eight algorithms which are Setup, Credential, Vote, Validate, Box, VerifyVote, Tally and Verify.

During the setup phase the public and private keys for the election are created and the list of credentials is also initialized empty. Then for each voter, a public and private key is also generated and adds the public key to the list of credentials thus ending the Credential phase.

In the vote phase the vote is encrypted using the election public key and a proof is computed to prove that the vote is valid. A signature is then made on the ciphertext and proof using the voter secret key. The ballot is then submitted as the voters public key, the encrypted vote and proof and the signature of the two previous parameters.

The validate phase is used to check if the vote is valid. It first checks if the public key is in the credentials list, the proof created during the voting phase is verified and so is the signature. The Box phase is when the vote is submitted in the bulletin board. This only happens if the voter authenticated successfully with an id. In case a vote is already present with the same id or public key as the vote that is about to be posted then the previous vote is deleted and the new vote is posted. If there is no such vote then the new vote is simply posted on the bulletin board. The votes are discarded in case the public key of the voter is not on the credentials list, if the vote did not pass the validate algorithm or if the ciphertext of the vote is the same as the previous one. The VerifyVote is used to check if the vote is present in the bulletin board.

In the tallying phase, the validation algorithm is passed on every ballot again and the voters public credential is also checked to see if it is in the credentials list. If this fails the election outputs an empty result. After this validation, the tally is computed relying on the homomorphic properties of the El Gamal encryption where the result should be in an interval between zero and the number of legitimate voters. Finally a proof is created. These results are validated in the Verify algorithm which does the validation of the previous phase again, creates the ciphertext and check the validity of the proof created before.

2.3 Discussion of suitability

Every e-voting protocol described in section 2.2 has both advantages and disadvantages in relation to the requirements that were described in section 1.2. A more concise version of these advantages and disadvantages is given in table 2.1.

In the case of Scantegrity II and Pret à Voter, they rely on the person being present at the location where the voting is taking place. For anonymization both these protocols make use of mix network, which require a number of servers and good management of the distributed resources. Although these systems have these disadvantages they provide an easy way to audit ballots and to verify that the vote was counted. Remoteegrity being an extension to Scantegrity II would force the implementation of all the Scantegrity II backend and as such force the use of mix networks whose disadvantages are mentioned above. It also uses the Scantegrity II ballots and another special code sheet which must be delivered by third party which could compromise the voter privacy in case someone got access to these beforehand.

EVIV provides a great way of maintaining user privacy but at the cost of the user having to carry a device to maintain the token used for the encryption process of the ballot and vote. It has the advantage of using code voting which increases the usability of the system during the verification phase of the

Table 2.1: Summary of Advantages/Disadvantages of different protocols

Protocol	Advantage	Disadvantage
Scantegrity II	Distributed shared key for election key End-to-End verifiable	Presencial voting Use of mix networks
Remotegrity	Remote code voting to assure correctness End-to-End verifiable	Uses paper ballots sent by third party Extension of Scantegrity
Prêt À Voter	Easy vote verification End-to-End verifiable	Presencial voting Use of mix networks
Civitas	Fake credentials for coercion resistance	Use mix networks
VeryVote	Code voting End-to-End verifiable	Code sheet sent by a third party Use mix networks The server can know who voted for whom
EVIV	Protects user privacy in unsafe computer Uses code voting End-to-End verifiable	Token for e encryption on external device
TrustVote	Hybrid protocol End-to-End verifiable Allows vote revocation	Use of large number of servers Paper and electronic casting are separate
sElect	Automated Verification System End-to-End Verifiable	Use of mix networks User must trust the computer
Helios v1	Only needs a single server End-to-End Verifiable	Use of mix networks User must trust Helios server and computer Possibility of ballot stuffing Possibility of ballot replay attack
Helios v2/3	Same advantages as Helios v1 Use of homomorphic encryption Distributed election key among trustees	User must trust Helios server and computer Possibility of ballot stuffing Possibility of ballot replay attack
Helios-C	End-to-End verifiable Verifiable if only one server is dishonest Solves replay attacks and ballot stuffing	Uses more than one server

protocol as it is easier for the voter to simply check if the correct code is presented to him or not. Similarly VeryVote uses the same idea of code voting used in EVIV but the code sheet must be delivered through a third party means. This, like in the case of Remotegrity, can compromise user privacy. Another problem with this system is that the entity who creates the code sheets is the election server and as such, during the election it has enough information in order to associate a code sheet to a voter and as such it could discover who voted for whom.

Civitas is one of the first protocols to provide a remote e-voting protocol which is coercion resistant. For this it uses the idea of fake credentials that the voter can create with an algorithm which is given.

Although TrustVote is the only e-voting protocol on this list who is hybrid, it also suffers from it. One problem with the protocol is the large number of servers required in order to validate a ballot. In order to turn it hybrid, the protocol also forces the end of the e-voting vote casting phase before a voter can cast a paper voting, ultimately increasing the time of the election. Even so the protocol offers a good approach on how to turn a e-voting protocol to a hybrid e-voting protocol. It also provides end-to-end verifiability and allows for vote revocation which can diminish the probability of vote coercion being successful although it is still possible.

The protocol sElect introduced the idea of an automated verification system when the user would

check the results of the election. This increased the percentage of votes that were verified since previously not all users would verify that their vote was counted. Although this automatic verification brings advantages it also makes the need of the user trusting the device that is verifying the vote, in this case the voters computer.

Helios presents various advantages and disadvantages depending on the version in question. The first version of Helios provided an open-audit system with the use of a single server, but it had problems in which the voter had to trust the computer he was voting from and the Helios server. It was also possible to do ballot replay attacks where a dishonest voter could cast the same ballot as another voter, or for the Helios server to cast ballots to itself. It made use of mix networks which were then dropped in the next versions. The second and third version started using a homomorphic encryption scheme, thus allowing anonymization without the use of extra servers and also started using distributed shared keys for the election key which made it impossible for only one person to decrypt the ballots. Although these issues were addressed there were still problems from the previous version that carried on, like the ballot stuffing. These versions of Helios also suffered from usability issues that are discussed by Fatih Karayumak et al. [36]. From the ballot stuffing problem Helios-C was created which introduced the credentials idea to the Helios system. This solution solved the ballot replay attacks and the ballot stuffing problem from previous versions of Helios and removed the single point of failure the system had, which was the Helios server being dishonest. The downside was the necessity of another server to create credentials for each voter. These protocols gave way to a protocol named Belenios. This protocol, which takes advantage of Helios simplicity and maintains Helios-C credential mechanism was the one chosen for the solution. This is due to maintaining advantages of both the protocols, except the threshold on the shared key generation present in Helios-C. Although this is a security risk, it also brought an advantage of simplifying the use of the system. A more detailed explanation of the protocol is presented in section 3.2.

2.4 Chapter Summary

This chapter started by explaining some ideas that are behind most of the current e-voting protocols. It then presents some core concepts which are necessary for the reader in order to understand and reflect on e-voting protocols. Following this, a number of e-voting protocols are explained, ranging from in-person e-voting protocols, to remote e-voting protocols and even an example of a protocol which adapts an existing e-voting protocol to turn it remote. An explanation of how these protocols work is given, and in the end, a reflection on each protocol is done in order to determine which would be best to adapt to a hybrid protocol and be used in an academic environment.

Chapter 3

Protocols

This chapter starts by presenting the baseline protocols that were used in order to design the solution to the problem this thesis tackles. The protocols used were the already existing paper voting protocol, which is currently used in Instituto Superior Técnico, and the Belenios protocol, which is a protocol inspired by Helios and Helios-C. The assumptions and security properties of each of these baseline protocols is also presented. Afterwards the hybrid protocol H-Belenios, which is based on the previous two, is presented by exposing the specification of this. Next proofs of correctness, which are based on the assumptions and security properties of the baseline protocols, are presented in order to prove that the changes made on the protocol don't affect the security of this.

This information was also published in Actas do 9º Simpósio de Informática, Inforum 2017 [7].

3.1 Baseline Protocol A: Current Paper Voting protocol

Currently at IST, voting is still done in a classical paper format. This follows a specific protocol which is well documented. Although some changes are performed on the protocol depending on the type of election which is occurring, this thesis focuses on the specification of a general election. For such, research was made on the document which presents the rules and regulations of such protocol [5].

The main entities involved in an election done using this protocol are the electoral commission, the poll workers, which are members present at each table, and each eligible voter.

The current paper voting protocol can be divided into four main phases:

Set up phase. Before the election begins, the members of the electoral commission, including the president, are designated. These members cannot be candidates for the election and include, in particular, one member per election candidate which serves as a representative. On the day of the election, one or more voting tables are available for depositing votes. Each voting table has a group of poll workers (which may include one representative per candidate) designated to it.

Voting phase. In order to vote, the voter first shows his identity to the voting table president, who also verifies his voting right. The voting secretary notes on the electoral roll that this voter has voted

and the president hands out a paper ballot. The voter goes to a voting booth, and fills in the ballot secretly by placing an 'X' next to the candidate of choice. Any other type of mark on the ballot invalidates it. Finally, the voter hands in the folded ballot to a person present at the voting table who inserts it into the ballot box.

Tally phase. At the end of the election the pool workers at each table proceed to count the votes and their distribution. The minutes are then written, recording the information gathered from the votes. The partial results are then handed to the electoral commission who will add all the votes from different tables and calculate the tally. The results must be released 24 hours after the end of the voting phase, and the paper votes must be destroyed 30 days after the election has passed.

Audit phase. After the tally phase complaints can be made to the electoral commission with a time limit of one day after the results are published. Every pool worker also has the option of complaining in the table report against decisions made by that table.

From this protocol it is possible to conclude the following security properties and assumptions:

Assumptions:

1. No one, including the pool workers, interferes with the paper votes in the ballot boxes.
2. The pool workers perform the tally and publication of the results according to regulations.

Security Properties:

1. Only and all eligible voters are able to vote.
2. Each vote is kept confidential (regarding both existence of vote and vote content, for a given voter), up to aggregation of results.
3. The regulations specify a correct procedure for counting exactly one vote per voter.
4. It is possible to retally the votes, within a specified time frame.

3.2 Baseline Protocol B: Belenios

Belenios¹ [6] is an e-voting protocol based on the Helios and Helios-C protocols. This protocol takes advantage of the simplicity which is present in the Helios system while adding the advantages brought by of using credentials in Helios-C. This protocol however does not use a threshold for the key generation protocol used for generating the election public key by the trustees.

In order to better understand the protocol, it is necessary to first know who are the main entities which are going to interact with it. These are:

- Voting Server where all the main operations regarding the election happen.

¹<http://belenios.gforge.inria.fr/>

- Credential Authority whose main objective is to generate and distribute the credentials for the voters of the election.
- The Server Administrator who is the person who created and manages the election.
- The Voters who are the users who are eligible to vote.
- The Trustees who are persons who are trusted and as such are able to keep a share of the secret key that at the end of the election is used to decrypt the encrypted tally.

Knowing the entities which participate in this protocol, it is possible to divided it into four main phases:

Set up phase. An election in Belenios starts with the creation of the credentials. The server administrator sends a unique universal identifier (uuid) to the credential authority. There, a list of credentials and their public parts is created. Each credential is sent to the respective voter, and the list of shuffled public parts for the credentials is generated and sent to the server administrator.

Then, each trustee generates her own public key and sends it to the server administrator, who verifies that the trustee has the secret part of it, and in the end combines all of the trustees keys in order to generate the election public key. With both the list of public credentials and the election public key, the server administrator is able to create the election. The protocol uses a public key encryption scheme with additive homomorphic properties.

Voting phase. In order to vote, the voter obtains the election parameters, creates a ballot which is formed from the encrypted votes and proofs (proof of membership and proof that the vote is formed correctly), the signature and the election uuid, and obtains a hash of the ballot. If all the data present in the ballot is valid then the voting server publishes it.

Tally phase. The election is terminated by the server administrator. The homomorphic properties of the encryption scheme are used to aggregate all the votes into an encrypted tally which is sent to each trustee to perform a partial decryption. Finally, the server administrator verifies the partial decryptions and aggregates them in order to create the decrypted tally, which is then published.

Audit phase. During the voting phase and tally phase, each voter is able to verify that their vote was cast correctly using the smart ballot tracker (hash of the serialization of the ballot) which was published on the bulletin board when the ballot was cast. It is also possible to verify that the encrypted tally was calculated correctly. The greater the number of voters performing this verification, then stronger is the confidence in the outcome of the election.

From this protocol the following security properties and assumptions can be extracted:

Assumptions:

1. Not both the voting server and the credential authority are simultaneously corrupt.
2. The client software does not leak information about the electronic votes.
3. The trustees will not work together in a malicious way.

Security properties:

1. Only and all eligible voters are able to vote.
2. Each vote is kept confidential (regarding vote content), up to aggregation of results.
3. The protocol specifies a correct procedure for counting exactly one vote per voter.
4. Each voter is able to verify that their vote was cast correctly.
5. Each voter is able to verify that the tally is correct.

Although Belenios allows for the recovery of credentials, it was considered outside the scope of this thesis and as such it is not presented. Another version of Belenios was proposed, called BeleniosRF [37] which proposed a system that allowed for a more efficient coercion resistance.

3.3 H-Belenios

H-Belenios is the protocol which was designed in order to solve the problem presented in this thesis. It is a hybrid protocol based on the the two baseline protocols presented in section 3.1 and section 3.2. This protocol tries to minimize the changes done on the baseline protocol A so that users who are less technical experienced or users who simply don't feel comfortable using an electronic voting protocol can still vote without much change. This protocol also allows for revocation of the electronic vote by casting a paper vote afterwards, since this last always takes precedence over his electronic counterpart.

The parties that participate in the hybrid protocol include those of BP-B – the voting server (**VS**), the credential authority (**CA**), the server administrator (**SA**), each eligible voter (**V**) and each trustee (**T**) –, as well as that which is present in the paper part of the protocol – the poll worker (**PW**). In order to formally describe the integration of the two baseline protocols, it is necessary to make explicit the different structures where data is saved: The **EL**, which is a list of all the elections, including their parameters (dates, descriptions, names) and their uuid; the eligible voter list **EVL**, which maps each voter identifier to their respective public credential; the list **TL** of trustees; the table of electronic votes **TEV** which contain all the e-votes done by eligible voters; the list of identifiers of paper voters **LPV**, that is the representation of the paper voters on the system and the table of final electronic votes **TFEV**. There is also **pLPV** which is the same as **LPV** but only in paper format, as is currently used in BP-A. From these structures, during the voting phase the smart ballot tracker of each ballot in **TEV** will be published and after the tally the smart ballot tracker of each ballot in **TFEV** will be made available together with the voter identifiers of those who voted by paper. The data structures which are made public will be available to view in the smart bulletin board **SBB**.

In terms of security schemes used by the protocol, it maintains the use of exponential El Gamal for the encryption of the ballots. This is due to the additive homomorphic property it possesses. As for the signature scheme used in the ballots, it was changed for the Schnorr signature scheme, which also uses the same parameters has the exponential El Gamal and produces short signatures.

In order to minimize redundancy, this thesis focuses more on the differences of the protocol. The election process can be divided in four phases. Each of these phases is described bellow in more detail accompanied by a table that corresponds to the formal description. Finally, it is assumed that all message exchanges are being done over secure channels.

3.3.1 Set up Phase

This is the first phase of the protocol where the election is prepared. Similarly as in Belenios, the **SA** must first send to the voting server the election parameters (eParam). These parameters are the name of the election, the description of the election, the start and end dates and if applicable the start and end times for the ballot box (these determine the time interval in which a voter is allowed to cast a vote during the voting phase). As the server receives this information it generates a uuid for the election, saves this information to **EL**, publishes this information on **SBB** (this is denoted using the publish), and generates the cryptographic parameters necessary for the election (two primes p and q where $p - 1 \equiv 0(mod q)$, and g which is a generator for Z_p).

The **SA** must also send **TL** to the voting server in order to start the key generation process. After this **T** can generate their own El Gamal key pair. The private key is kept secret by the trustee and the public key is sent to the server with a proof of knowledge of discrete logarithm. Each of the keys is publish on **SBB** in order for everyone to verify that the aggregation is done correctly. After all **T** have sent their key shares to the server, the administrator can verify each of the zero knowledge proofs (denoted as validate) and aggregate the keys of the trustees generating the election public key which will then be used to encrypt the ballots. This key is sent to the server and published on **SBB** in order to be verified by anyone who wishes it.

For the paper part of the protocol, the set up phase happens as it did previously without any change.

Table 3.1: Set up phase specification

#	Message Exchange	Voting Server & Administrator
1	SA → VS : eParam	EL := EL ∪ {⟨ e_{id} , eParam⟩} publish(⟨ e_{id} , eParam⟩)
2		
3		
4	SA → VS : TL	publish(Pk_t), $\forall t \in \mathbf{TL}$
5	$t \rightarrow \mathbf{VS} : Pk_t, \delta_t, \forall t \in \mathbf{TL}$	
6		validate(δ_t), $\forall t \in \mathbf{TL}$
7	VS → SA : $Pk_t, \delta_t, \forall t \in \mathbf{TL}$	
8		publish(Pk_e)
9	SA → VS : $Pk_e = \sum_{t \in \mathbf{TL}} Pk_t$	
10		
11	SA → VS : EVL	

3.3.2 Voting Phase

The second phase of the protocol is the voting phase which occurs in during the dates defined by **SA** while creating the election. During this phase any $v \in \mathbf{EVL}$ can cast a vote either by paper, electronically or both (paper votes have precedence). In case the voter wishes to vote electronically he must first

register himself with the credential authority. To do so, he sends a request to v who then send to **CA** the identifier of v and the cryptographic parameters generated during the set up phase. **CA** then generates the credentials which are a key pair to be used in the schnorr signature scheme. These credentials are then sent to v by email and the public part of the credentials is sent to **VS**. After receiving his credentials, v will have access to the ballot box. Here he must insert his private and public credentials, choose his answers for the election and creates the ballot. The ballot is composed of the encrypted answers (done with the election public key), a digital signature done with the private credential, individual zero knowledge proofs that each encrypted answer belongs to a finite and proof that the overall answer also belongs to a finite set. As the ballot is formed, a hash of the ballot is computed, which corresponds to the smart ballot tracker, and v saves this. Finally v can cast his ballot to **VS**, which verifies the zero knowledge proofs, saves the ballot to **TEV** and calculates the hash over the ballot and publishes it on **SBB**.

Table 3.2: Voting phase specification for electronic voting

#	Message Exchange	Voting Server
1	$v \rightarrow \mathbf{VS} : Id_v, \text{request}$	$\mathbf{EVL} := \mathbf{EVL}[Id_v \mapsto Pc_v]$ $\text{validate}(Pc_v \notin \mathbf{TEV})$ $\mathbf{TEV} := \mathbf{TEV} \cup \{(Pc_v, \text{eBallot})\}$ $\text{publish}(\mathbf{TEV})$
2	$\mathbf{VS} \rightarrow \mathbf{CA} : Id_v, p, q, g$	
3	$\mathbf{CA} \rightarrow \mathbf{VS} : Pc_v, Id_v$	
4		
5	$\mathbf{CA} \rightarrow v : Pc_v, Sc_v$	
6	$\mathbf{VS} \rightarrow v : \text{eParam}, p, q, g, Pk_e$	
7	$v \rightarrow \mathbf{VS} : \text{eBallot} = ans, s\{ans\}_{Sc_v}, \delta_{ans}, Pc_v$	
8		
9		
10		

In case v wishes to vote by paper he may do so as it currently occurs. He goes to a poll booth to vote, proves his identity to **PW** and votes by paper. The only difference happens at the end of this phase, in which **PW** must give to **SA** **pLPV** who then submits the electronic version **LPV** to **VS**.

Table 3.3: Voting phase specification for paper voting

#	Message Exchange	Server & Administrator	Voting Table
1	$v \rightarrow \mathbf{PW} : Id_v$	$\mathbf{pLPV} := \mathbf{pLPV} \cup \{Id_v\}$ $\mathbf{LPV} := \mathbf{pLPV}$	$\text{validate}(Id_v \in \text{dom}(\mathbf{EVL}) \setminus \mathbf{pLPV})$ $\mathbf{BB} := \mathbf{BB} \cup \{\text{pBallot}_{id}\}$
2			
3	$v \rightarrow \mathbf{PW} : \text{pBallot}$		
4			
5			
6	$\mathbf{PW} \rightarrow \mathbf{SA} : \mathbf{pLPV}$		
7			
8	$\mathbf{SA} \rightarrow \mathbf{VS} : \mathbf{LPV}$		

3.3.3 Tally Phase

After the voting phase ends, then begins the tally phase. **VS** calculates **TFEV** which is the table of final electronic votes. On this table are present all of the votes of voters who didn't cast a paper vote. **VS** is able to identify these from the public credentials which are associated with both a ballot and the

voter. These votes are then aggregated using the additive homomorphic properties of exponential el Gamal. Following this, **T** must partial decrypt this aggregation and send it to **VS** with a zero knowledge proof of correct decryption. **SA** can verify these proofs and aggregate the partial decryption forming the electronic tally which he send to **VS**. **SA** also submits the tally of paper ballots to **VS**. After all the verifications are done **VS** can publish the results of **SBB**.

Table 3.4: Tally phase specification

#	Message Exchange	Server & Administrator	Voting Table
1		$\mathbf{TFEV} := \mathbf{TEV}^{\mathbf{EVL}(\mathbf{LPV})}$	
2		$\mathbf{eRes} := \text{tally}(\mathbf{TFEV})$	
3			$\mathbf{pRes} := \text{tally}(\mathbf{BB})$
4	$\mathbf{VS} \rightarrow t : \mathbf{eRes}, \forall t \in \mathbf{TL}$		
5	$x \rightarrow \mathbf{VS} : D_t\{\mathbf{eRes}\}, \delta_{D_t}, \forall t \in \mathbf{TL}$		
6	$\mathbf{VS} \rightarrow \mathbf{SA} : D_t\{\mathbf{eRes}\}, \delta_{D_t}, \forall t \in \mathbf{TL}$		
7		$\text{validate}(\delta_{D_t}), \forall t \in \mathbf{TL}$	
8	$\mathbf{SA} \rightarrow \mathbf{VS} : D\{\mathbf{eRes}\}$		
9	$\mathbf{SA} \rightarrow \mathbf{VS} : \mathbf{pRes}$		
10		$\mathbf{fRes} := D\{\mathbf{eRes}\} + \mathbf{pRes}$	
11		$\text{publish}(\mathbf{TFEV}, \mathbf{LPV})$	
12		$\text{publish}(\mathbf{fRes})$	

3.3.4 Audit Phase

The audit phase occurs in parallel to the phases described above and in a similar fashion as in Belenios. At the end of the election all the voters will be able to check that their smart ballot tracker is present in **SBB**. For those voters who voted both by paper and electronically, since the vote that is counted is the paper one, then their smart ballot tracker will not be present there, instead appears a note saying that they voted by paper.

A checksum is also added in order to verify that the total number of votes is equal to the number of valid smart ballot trackers and the number of votes done by paper (given by $|\mathbf{finalResult}| == |\mathbf{TFEV}| + |\mathbf{LPV}|$). This enables to detect any mismatch between the number of counted paper votes and those that appear in the smart ballot tracker.

During the key generation process, the ballot casting and the decryption process, zero knowledge proofs are also used and verified in order to provide assurance of the data being submitted, which are the same as the ones used in Belenios.

3.4 H-Belenios Correctness

Some relevant security properties for H-Belenios are now presented. First it is necessary to express what are the assumptions and definitions which are considered.

Assumptions: The union of those enunciated for the baseline protocols A (BP-A) and B (BP-B).

Definition 1. *Final vote of an eligible voter*

The final vote of an eligible voter is either the paper vote (if the voter cast a paper vote), the last vote cast electronically (if the voter did not cast a paper vote), or undefined (if the voter did not vote).

- *Not defined, if the voter did not vote.*
- *The vote expressed on the paper vote, if the voter issued a paper vote.*
- *The last vote issued electronically, otherwise.*

Property 1. *Only and all eligible voters are able to vote.*

Proof.

- All eligible voters can vote. If an eligible voter wants to vote by paper he can do so as in BP-A. If a voter wishes to vote electronically he can do so, as in BP-B, from any web browser so long as he registers to vote electronically and authenticates himself in the process.
- Only eligible voters can vote. As in BP-A, a voter has to prove his identity in order to vote by paper. If a voter wants to vote electronically, he must register as such using a web browser, as in BP-B. To do so he must authenticate himself using his authentication credentials and to cast a vote he must be authenticated and have both credentials that are sent to him when he registers himself to vote.

□

Property 2 (Confidentiality). *Each vote is kept confidential (regarding vote content), up to aggregation of results.*

Proof.

Confidentiality (regarding vote content) of votes up to publication of the results is guaranteed for paper votes in the same manner as for BP-A, and for electronic votes as for BP-B. The publication of **LPV**, as well as the construction of **TFEV** does not reveal vote content.

□

Property 3 (Integrity). *The protocol specifies a correct procedure for counting exactly one vote per voter.*

Proof.

1. The protocol specifies a correct procedure for building a list of paper voters **LPV** and a table of final electronic votes **TFEV**.

Indeed, from BP-A it is possible to conclude that **LPV** is correctly built, since this is not altered in any way. Each time a voter votes electronically, his ballot is added in **TEV** (line 9 in table 3.2). These ballots have a connection to the ids of the voters via the public credential. Finally, **TFEV** is constructed from **TEV** and **LPV** (line 1 in table 3.4) by removing from **TEV** the ballots cast from voters who have their ids in **LPV**. Given that **LPV** and **TEV** are correct, it is possible to conclude that **TFEV** is also correct.

2. The protocol specifies a correct procedure for tallying tables **LPV**, **TEV** and **TFEV**.

3. All and only final votes are counted.

- Only final votes are counted:

According to line 10 of the table 3.4 the final result is composed from the decryption of eRes, which is the aggregated encrypted result, and pRes. pRes is the tally of the paper votes, calculated from the votes present in the physical ballot boxes as in BP-A. The tally of the BBs counts exactly one vote per voter, as in BP-A.

- From the definition of final vote it is possible to conclude that a final vote is either in paper or in electronic format, but in only one of these formats.
- If the final vote is in paper, then according BP-A, it is assumed that the vote is counted and that according to the correctness of TFEV, if an electronic vote from the same voter was cast, it will not be counted.
- If the final vote is electronic, from the correctness of the TFEV it can be concluded that it will be counted.

- All final votes are counted:

According to definition 1, a final vote is either in paper form or in electronic format. If the vote is in paper form then according BP-A, it is assumed that the vote is counted correctly. If the vote is electronic then, according to correctness of **TFEV**, it is counted correctly.

- It is defined in the the specification, that the number of cast votes, paper and electronic, are added and compare to the number of paper voters and the number of final electronic votes which corresponds to the number of electronic voters who did not cast a paper vote. These results need to be equal in order for the election to be valid and to prove that each final vote was counted at most once.

4. An eligible voter will have exactly one counted vote if and only if he has issued at least one vote.

- (a) If an eligible voter issued a vote, he will have a final vote. Indeed, if he issued a paper vote, this will be his final vote. Otherwise, the vote is electronic, in which case it will also be a final vote.
- (b) Each eligible voter will have at most one final vote. Indeed, as with BP-A, a voter cannot cast a paper vote more than once. By definition of final vote, even if a voter has both paper and electronic votes, then only the paper vote is the one that is considered for the election.

□

Property 4 (Auditability). *It is possible to retally the paper votes, within a specified time frame. Furthermore, assuming that the tally of the paper votes is correct, every voter is able to verify that the final tally is correct.*

Proof.

The partial tallies (paper and electronic votes) can be verified as in BP-A and B. The combination of both enables to verify the final tally.

□

Coercion resistance. While BP-A is considered to be strongly coercion resistant, BP-B suffers from the fact that vote receipts can be produced. Although this is assumed to not be of primary concern in the context of this work, it is possible to observe that as a result of the integration of the protocol with a paper-based voting system, the new protocol improves on the level of coercion resistance that is provided by BP-B. Indeed, the voter now has a choice of also casting a paper ballot, which is preferred over the electronic ballot during the tally. As such there is no way to prove who he voted for, up to aggregation of the results.

Relaxing assumptions. The integrity properties of BP-A rely on strong assumptions of correctness of the actions of the poll workers. However, these assumptions can be relaxed in face of the new possibility of detecting a mismatch between the counted number of paper votes and the electronic votes published in the smart bulletin board. More precisely, it is no longer necessary to assume that fake ballots or votes cannot be added to or removed from the ballot boxes or tally, but only that they are not changed.

3.5 Chapter Summary

This chapter started by presenting the current paper voting protocol and the Belenios e-voting protocol which served as baseline for the solution. These protocols are thus explained and a list of assumptions and security properties are presented.

Having understood these protocols, this thesis solution is presented. H-Belenios is a hybrid protocol that makes use of the advantages of both the baseline protocols. A detailed specification of the protocol is presented each phase (except the audit phase) is accompanied by a table which represents it. It is then proven, according to the assumptions and security properties of the two baseline protocols that the changes done on the protocols don't affect security and in some cases even improves on the assumptions of the paper voting and the problem of vote coercion of Belenios.

Chapter 4

Implementation

This section gives details on the implementation. Through this, it is expected that the reader can gain some knowledge on the technologies used and some decisions that were done. The architecture that was used during development and testing, and what should be changed in it in order for the system to be deployed shown. This chapter also specifies how some of the more troubling situations were done in the back-end. Following this is a bit of details on the cryptography used and a description of the front-end of the application.

This chapter ends by showing a more close relation between the specification and implementation.

All the code of the implementation and its comments will be available at the github project page [8]. A wiki will also be in development in parallel which will contain information regarding the project and a link to this document.

4.1 Architecture

The architecture of the implemented system is present in figure 4.1. It is possible to see that the voter, the server administrator and the trustees all use the client application in order to communicate with **VS**. The **VS** and the **CA** also communicate between each other in order to send the request of credentials to **CA** and to receive the public credential from this last. The **CA** also has a connection to a email server in order to be possible to send the credentials. No secure channels were used since this architecture was only used for the development and for a closed network evaluation, and as such it should not be used in a real world deployment.

In a real world deployment, the connections done between the voter, **SA** and the trustees to **VS** should all use a form of secure channels such as TLS. **CA** should also remain hidden behind a firewall. The firewall should be configured to only allow input messages from **VS** and the email server (so that it can receive the acknowledgements that the emails were sent) and should only allow output messages to these two.

It is also assumed that the browser in which the client application is running is considered secure.

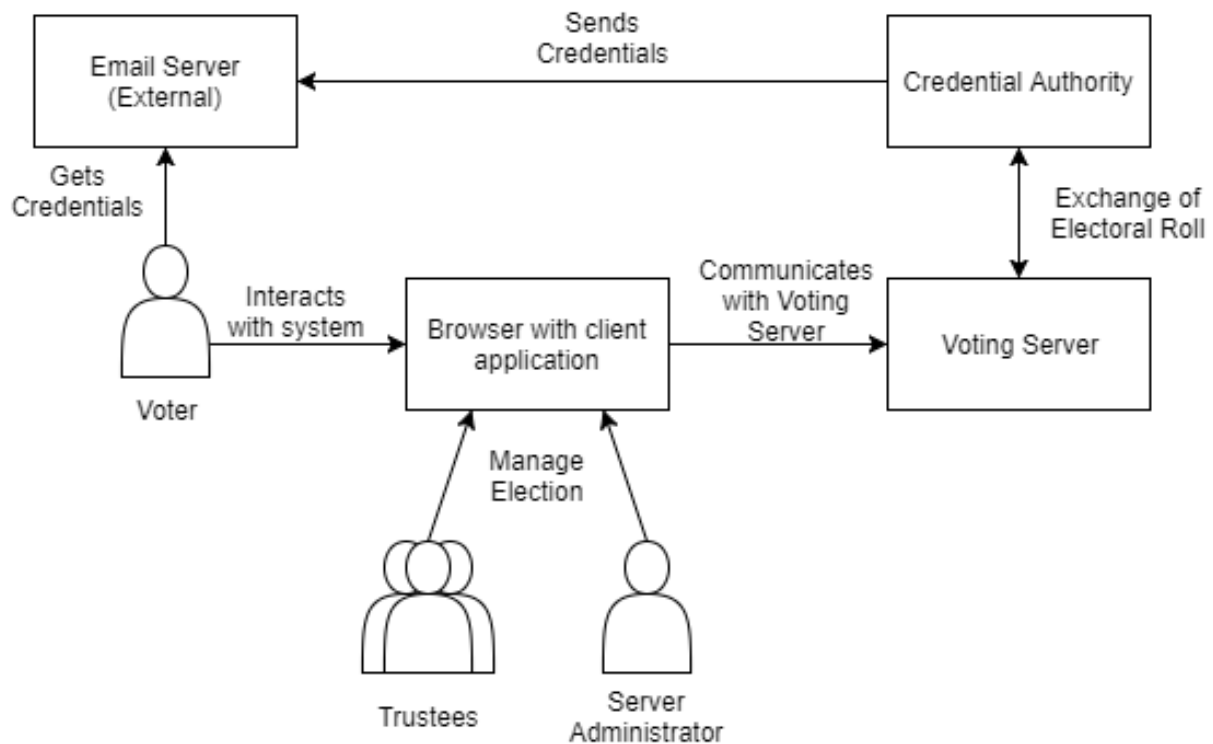


Figure 4.1: Architecture used

4.2 Adopted Technologies

During the implementation of this thesis, the technologies that were proposed to be used were the Java [38], Python [39] and NodeJS [40]. NodeJs, although it has gain popularity over the last years and has a high number of modules that extend the its functionality has the disadvantage of having low precision with very high numbers. Between Python and Java, the first was chosen due to it's versitility, high number of modules and easy to use frameworks. Another reason for choosing Python over Java was that currently, web browser are starting to deprecate the use of Java Servlets and as such, Python was seen as the best solution.

With the programming language chosen, the next step was to chose the framework in which to work. This was the Django framework [41] which allows the creation of web application in an easy and fast way. It provides many modules which automatically take care of some troubles when dealing with web applications, like authentication, sessions, injection and other. An external module chosen for the Django framework was the Python Social Auth [42] which allows to connect via OAuth or OAuth2 [43] to different web services. Unfortunately it did not provide the necessary back-end modules to connect off the box to the Fenix framework and as such it was necessary to implement these. More details on this are present in section 4.3.2.

For the front-end of the application, web languages such as HTML, CSS and JavaScript were used. Since the html was rendered in the back-end using the Django framework, it was possible to use Django templates in order to provide more dynamic pages for the users. JQuery was also used in order to provide more flexibility to JavaScript and some extra libraries were needed for it in order to provide more

functionality. These were:

- Tom Wu's Big Integer Implementation [44].
- vkThread plugin which allows to use threads in JavaScript [45].
- Stanford JavaScript Crypto Library [46].

4.3 Back-end Details

In this section, some details on the implementation of the back-end are going to be presented. These details include how the database is composed, how the connection to the Fenix framework happens, how the cryptographic parameters are generated and the administrator functions that are currently implemented. The settings for most of these details are present in the "settings.py" file of each of the applications. These files have all the configuration on how the system is going to run, which modules it is going to use, what database is configured and other.

One important detail to take note is that the connection between the **CA** and the **VS** is defined in the settings.py of the **VS**. There is present a variable called `CREDENTIAL_AUTHORITY` which saves the address of the **CA**. This is so that this address is accessible from every part of the application if needed. Another implementation detail on the **VS** settings.py file is the use of a variable called `AUTHORIZED_ADMINS`. This is a list of all the identifiers of users who are allowed to create elections. The original idea was to use the organizational status of the user (is the user a professor? a staff member? a student?) who logged in to the application and based on the status, it would allow the person to create an election or not. This idea was discarded because more status could be added in the future or changed. As such, as a current temporary solution, all ids of users who are allowed to create elections should be present in that list. In the future, if access to LDAP is granted, it can be used in order to determine who is allowed to create an election or not.

4.3.1 Database

Django uses object-relational mapping in order to save objects in the database. In figure 4.2, it is possible to see how all the objects relate to each other and how they are represented in the database of the **VS**.

A more detailed description of each model is presented in the following list. There a description on the purpose of the model is followed by a list of fields and what they information they save.

FenixUser This model stores the information regarding an user of the system. The information stored is fetched from the Fenix API using the OAuth authentication. This is described in more detail in section 4.3.2. This model extends the Abstract User model from Django and is used to substitute the default user model. The system can be configured to use other user models by changing the field `AUTH_USER_MODEL` present in the settings.py file of the Voting Server.

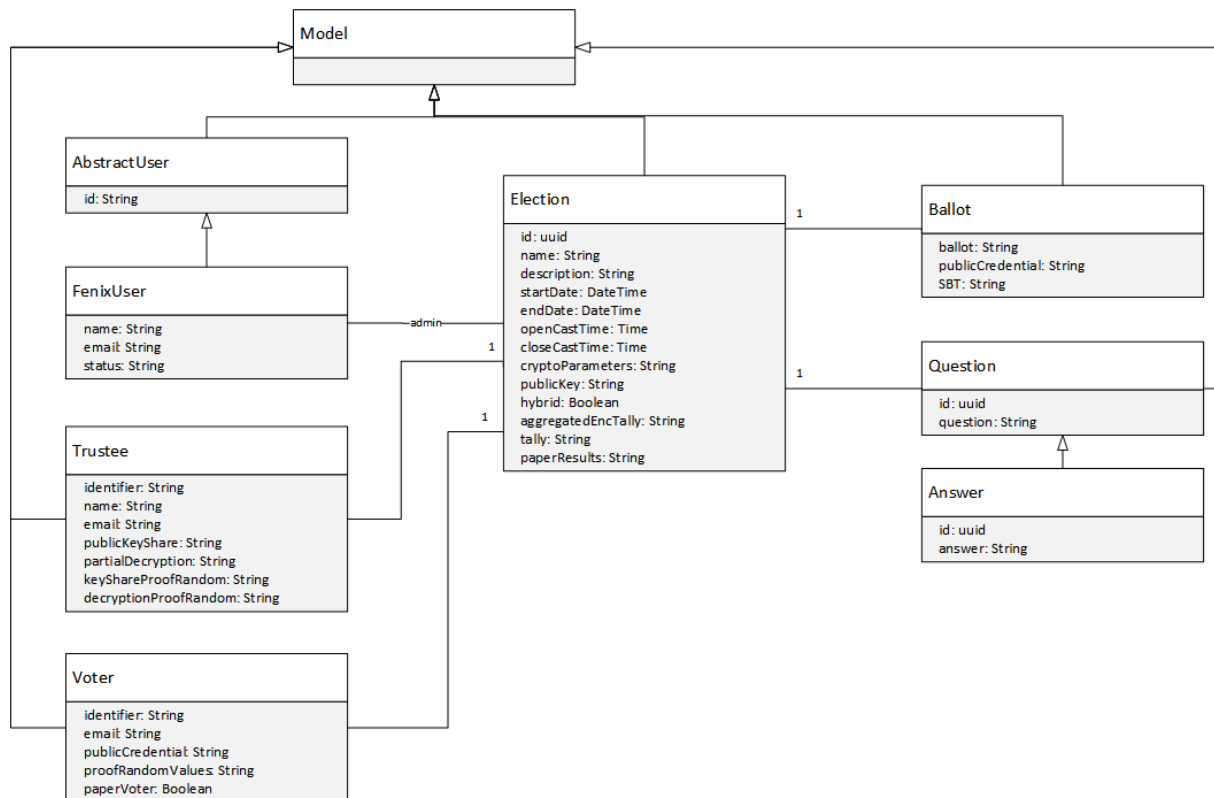


Figure 4.2: Models used

- id, is the primary key of the model and saves the Fenix identifier of the user. This field is inherited by the Abstract User model.
- name, is the name of the user.
- email, corresponds to the email of the user which is registered in Fenix.
- status, corresponds to the status the user has inside Fenix. This can be student, teacher, staff or other. Although currently not used, it is saved for easier access in a future improvement.

Election This model stores the information for the election. It serves as a center point for all other models which belong to this election to connect to.

- id, is the primary key of the election. This id corresponds to an uuid version 4 (the uuid is generated randomly instead of following a specific algorithm like version 1). It is used to identify the election.
- name, is the name of the election given by **SA**.
- description, is the description of the election. This field can contain information regarding the election which the **SA** may think is relevant.
- startDate, is a DateTime object which saves the start date and time of the election.
- endDate, similar to the startDate field, except it saves the end date and time of the election.
- openCastTime, is a Time field which saves the time at which an eligible voter may start to vote. It is used to limit the time limit in which the ballot box are open. In case no time limit is

defined, this value will be empty.

- closeCastTime, similar to openCastTime except it stores the time at which the ballot box closes for the day. It then reopens the next day at the time defined in openCastTime.
- cryptoParameters, stores the cryptographic parameters used during the election in json format. More details on how this parameters are generated is present in section ??.
- publicKey, stores the public key which was generated for the election.
- hybrid, stores a boolean which informs if the election is occurring in hybrid mode or not. This allows the system to be usable in case the **SA** does not wish to have paper votes for the election.
- aggregatedEncTally, saves the encrypted result of aggregating all the final electronic ballots. This information is store in json format.
- tally, saves the final result of the election in json format.
- paperResults, saves the result of the paper voting in json format. If the election was done without the hybrid flag, then this field will be empty.

Trustee This model stores all the information regarding a trustee for a specific election.

- identifier, represents the trustees Fenix identifier (istXXXXXX)
- name, stores the trustees name.
- email, stores the trustees email. Although it is currently not used, it is stored for in a future iteration of the system to send email with notification to the trustees.
- publicKeyShare, stores the information regarding the trustees public key share in json format. This information includes the trustees public key share and the proof that the keys were generated correctly.
- keyShareProofRandom, stores the random number used for the generation of the proof that the key share was generated correctly.
- partialDecryption, stores the partial decryption done on the aggregated encrypted tally in json format. It also stores a proof that the decryption was done correctly.
- decryptionProofRandom, stores the random number used for generating the proof that the partial decryption was done correctly.
- election, is the foreign key which connects this trustee to the election to which he belongs to.

A restriction was added on this model in order to prevent the same trustee from being present more than once in a specific election. As such the restriction added was an unique together restriction, in which the values of the fields identifier and election may not appear together more than once.

Voter This model stores the information of an eligible voter for a specific election.

- identifier, stores the voter Fenix id (istXXXXXX).

- email, stores the voter email. This email is then used to send the credentials to the voter.
- publicCredential, stores the voter public credential after he has registered himself to vote electronically.
- proofRandomValues, stores in json format all the random numbers used for the generation of the proofs that are present in the ballot which verify if the vote is valid or not.
- paperVoter, stores a boolean which represents if this voter voted by paper or not. This is important in order to identify the voters who voted by paper and their public credentials, which then are used to identify the ballots to remove from the electronic tally.
- election, is a foreign key for the Election model which represents the election to which this voter is eligible.

A restriction was added on this model in order to prevent the same voter from being present more than once on the same election. The unique together restriction was used on the fields identifier and election.

Question This model represents a question done on an election.

- id, is the primary key and identifies the question. It is stored as an uuid version 4.
- question, stores the text of the question itself.
- election, is a foreign key which identifies to which election this question belongs to.

A restriction was added on this model to prevent the same question from appearing more than once on a specific election. This is an unique together restriction and is used on the fields question and election.

Answer This model represents an answer for a specific question.

- id, is the primary key and identifies the question. It is stored as an uuid version 4.
- answer, stores the text of the answer itself.
- question, is a foreign key which identifies to which question this answer belongs to.

A restriction was added on this model to prevent the same answer from appearing more than once for a specific question. This restriction is used on the fields answer and question.

Ballot This model stores the information regarding a ballot cast by a voter.

- ballot, saves the ballot in json format.
- publicCredential, stores the public credential of the voter who submitted this ballot.
- SBT, stores the smart ballot tracker calculated by the server.
- election, which is a foreign key to the election to which this ballot was cast to.

An unique together restriction was used on the publicCredential and election fields in order to avoid a voter submitting more than one ballot.

The **CA** also has a database in which two type of model are saved. These are the election model and the credential model which are described as following:

Election This model represents an election. The **CA** uses this model in order to identify which election has ended in more than 30 days so that it can delete the information respecting it.

- id, is the primary key and stores the election identifier.
- endDate, stores the DateTime object which has the date and time of the end of the election.

Credential This model saves the public credentials that were generated for a specific election. This is to avoid the possibility of collision of credentials on the same election.

- credential, stores the public credential generated.
- election, foreign key used to connect this credential to an Election model.

In order to avoid having the same credential appear more than once on an election it uses an unique together restriction.

The connections to the databases are configured in the settings.py file of each server. In order to change the database in use, it is only necessary to change the settings as seen in listing 4.1 and afterword to run the migrations command so that the corresponding sql and database is ready to receive data.

Listing 4.1: Example of settings for MYSQL database

```
1 DATABASES = {
2     'default': {
3         'ENGINE': 'django.db.backends.mysql',
4         'NAME': 'dbName',
5         'USER': 'dbUser',
6         'PASSWORD': 'dbPassword',
7         'HOST': 'dbHost',
8         'PORT': 'dbPort',
9     }
10 }
```

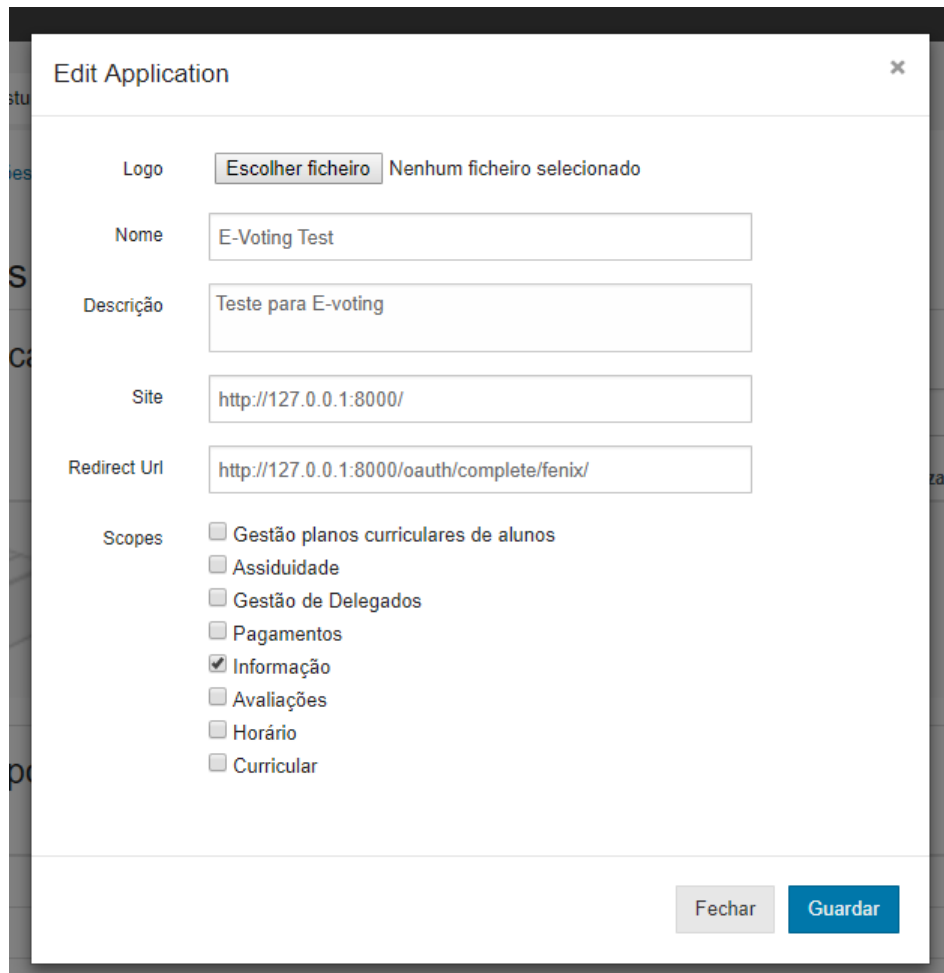
4.3.2 Connecting to Fenix

OAuth2 was used in order to connect to the Fenix authentication system. The Django framework already provides authentication mechanisms but out of the box it does not provide OAuth2 support. As such, the Python Social Auth module was used. This module provides mechanism in order to authenticate using OAuth and OAuth2 to various services and makes a connection with the Django authentication mechanism.

Although Python Social Auth has a number of back-ends to connect to various services, it did not have one to connect to Fenix. So, in order to use this module, a back-end was created. This back-end named FenixOAuth2, extends the BaseOAuth2 which is included with Python Social Auth, and provides

information regarding the authorization url, the access token url and other. It also has some functions defined in order to fetch the basic data from the server.

To use a custom back-end, it was necessary to define this in the settings.py file, under the AUTHENTICATION_BACKENDS field. It was also necessary to register the application in the Fenix system so that it could provide the client id and client secret. To do this, it is necessary to provide information from our application and the scopes of information it is expected to access (as seen in figure 4.3).



The image shows a web-based configuration window titled "Edit Application". It contains several input fields and a list of checkboxes. The "Nome" field is filled with "E-Voting Test", "Descrição" with "Teste para E-voting", "Site" with "http://127.0.0.1:8000/", and "Redirect Url" with "http://127.0.0.1:8000/oauth/complete/fenix/". Under the "Scopes" section, the "Informação" checkbox is selected. At the bottom right, there are two buttons: "Fechar" (grey) and "Guardar" (blue).

Figure 4.3: Fenix OAuth2 Authorization Configuration

Finally in order to access more information on those scopes, it was necessary to add another function to the authentication pipeline (list of functions that are run when authentication occurs). This function is used to fetch the user name and email.

4.3.3 Connecting to the mail Server

The credential authority needs to connect to a mail server in order to be able to send the credentials to the voters. In this case the mail server which the application connects to is the webmail server provided by Instituto Superior Técnico.

In order to connect to the email server, it is necessary to provide in the settings.py the information present in the listing 4.2.

Listing 4.2: Example of settings for connection to email

```

1 EMAIL_HOST = "mail.tecnico.ulisboa.pt"
2 EMAIL_PORT = 465
3 #EMAIL_USE_TLS = True
4 EMAIL_USE_SSL = True
5 EMAIL_HOST_USER = 'username'
6 EMAIL_HOST_PASSWORD = 'password'

```

The EMAIL_HOST corresponds to where the email server is hosted and the port corresponds to the port used to send the emails. The EMAIL_USE_TLS and EMAIL_USE_SSL are used to inform if the email server is able to use these technologies. Finally the EMAIL_HOST_USER and EMAIL_HOST_PASSWORD are credentials needed in order to access the host.

The emails are then sent as html, using the EmailMultiAlternatives object and editing the email content type as seen in listing 4.3. The from_email can be changed in order to be a no-reply email.

Listing 4.3: Send Email using Django

```

1 msg = EmailMultiAlternatives(subject, text_content, from_email, [to])
2 msg.attach_alternative(html_content, "text/html")
3 msg.send()

```

4.3.4 Administrator Functions

Since the data regarding the election should be deleted after 30 days of its end, an administrative command was created on each of the applications to do that. Django provides a mechanism to create administrative commands which are supposed to be used by the Server Administrators (not to be confused with the server administrator of the protocol).

These commands are defined as seen in listing 4.4.

Listing 4.4: Administrative Command

```

1 class Command(BaseCommand):
2     help = 'Deletes elections that have ended over 30 days ago'
3
4     def handle(self, *args, **options):
5         try:
6             elections = Election.objects.all()
7             for election in elections:
8                 if election.endDate < datetime.datetime.now()
9                     -datetime.timedelta(days=30):
10                     election.delete()
11         except Exception:
12             print("An error occurred")

```

This command serves to delete the election data after 30 days of its end and should be set to a cron task to be run everyday at a specific hour. Since all the models which contain information regarding the

election are set to be deleted as the main Election object is delete, it is only necessary to delete the main object.

4.4 Cryptographic Details

In this section some of the cryptographic details of the implementation are presented. Details are given on how the cryptographic parameters are generated, how the random number are generated in the front-end, the zero knowledge proofs and others.

4.4.1 Cryptographic Parameters Generation

As mentioned in section 3.3.1, the **VS** needs to generate some cryptographic parameters which are then used in the encryption and signature schemes. These parameters are of high importance because if they are generated incorrectly, it could lead to security issues, like information leak which could compromise voter privacy or ballot replay which could lead to incorrect election results.

These parameters are generated on the **VS** for each election that is created. The first parameters is a 160 bit prime number which is denoted as q . To generate this parameter, first a random number is generated using the secrets module from the python standard library. This module provides random number generators which are said to be cryptographically secure, or in other words, are able to provide secure random numbers to be used in cryptographic schemes. After generating this value, it is test to check if it is prime. The system uses the Miller-Rabin primality test which is run 11 times (accuracy value).

After generating a value number for q it is time to generate the value p . This value is closely related to q since we can calculate p using $p - 1 \equiv 0(mod q)$. After having also generated p and verified that it is indeed prime it is finally time to generate g . This value is a generator for the of the cyclic group with of order p . In order to generate this number, first a primitive root of p is discovered. This is a value that verifies $a^{\frac{p-1}{f}} = 1(mod p)$ where f is a factor of $p - 1$. After having calculated a primitive root, it is possible to calculate the last parameter as $g = a^{\frac{p-1}{q}} mod p$ where a is the primitive root.

These cryptographic parameters are then used in the Exponential El Gamal and the Schnorr signature scheme.

4.4.2 Random Number Generation on the Front-End

During the encryption process and signing processes, random number generation is also necessary. This brought a problem since these operations occurred in the front-end and JavaScript doesn't work well with very big numbers since it loses precision due to storing them as 64-bit floating points.

As such the Big Integer library from Tom Wu was used in order to do these operations. Another problem with JavaScript is that it is not prepared to do cryptographic operations and as such, it does not provide a built in random number generator which is considered cryptographically secure. In order

to generate the random numbers, the Web Crypto API [47] was used since it provided a secure way to generate said numbers and is present in most modern browsers.

For the generation of the Schnorr signatures, their verification and the creation of the Smart Ballot Trackers used for the verification of the integrity of the ballot, SHA256 hashes were used. In order to implement generate these, the Stanford JavaScript Crypto Library was used since it provided an implementation of these hashes. Although it also provides a random number generator, this needs to be feed entropy from external sources, like key presses and cursor movement. This poses a problem cause it may not be random enough to provide good entropy. Another problem is that since these cryptographic operations are CPU intensive, they block the UI and as such were executed on different threads using web worker technology (via VkThread plugin). These web workers are run in a separate environment and as such these environments does not have access to the UI thread which holds the values necessary for generating entropy.

4.4.3 Zero Knowledge Proofs

Zero knowledge proofs were used in order to prove that certain operations were done correctly without disclosing sensitive information regarding the operations itself. The zero knowledge proofs used were:

- Proof of knowledge of discrete logarithm, which is used to prove to the **SA** that the trustee has the secret key for the public key share he submitted.
- Proof of correct decryption, that is used to prove to the **SA** that the trustee knows the secret key for decryption a message encrypted with his public key(or part of it in this case).
- Proof that a discrete logarithm belongs to a finite set, which is used to prove to the **VS** that the ballot cast by a voter is valid. This proof is run for each individual answer of a question(individual proofs) and to the group of answers to a question(overall proof).

More information on these proofs is available on Some ZK security proofs for Belenios by Pierrick Gaudry [48].

4.5 Front-End Details

In this section, the pages of the system are described. When a user first enters the system he will be welcomed by the homepage. In order to access more of the system, the user will need to login in using his Fenix credentials. The user will then have a sidebar where he will be able to see a list of all elections and access their bulletin boards, a list of elections were he is **SA** which allows him to access the management page and the list of elections were he is a trustee that allows him to access the trustee page. This section is divided based on the user type and the pages he will most likely use.

4.5.1 Server Administrator Pages

If the user has the permissions set in the settings.py file, then he will have a "Create Election" button available to him (see figure 4.4).

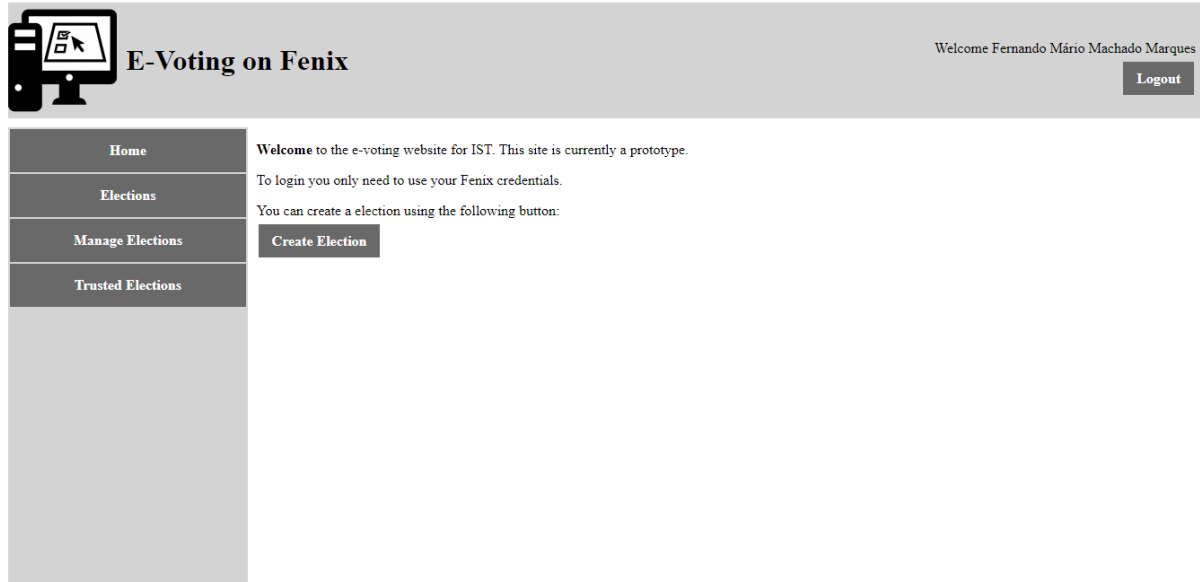


Figure 4.4: Homepage of the e-voting application

If the user clicks the "Create Election" button, he will be greeted by a form in order to input the election information (see figure 4.5). This information is the name, description, a value which informs the system if the election will be hybrid or not, the start and end dates and times for the elections (a widget is available to select these options) and the time of opening and closing of the election (the input texts used are enabled and disabled by a radio button which informs if there will be a time interval to submit ballots).

The screenshot shows the 'Create Election' form within the application. The form is titled 'Create Election' and is located in the main content area. It contains several input fields: 'Name:', 'Description:', 'Start Date:', 'End Date:', 'Start Cast Time:', and 'End Cast Time:'. There are also radio buttons for 'Will the election have paper votes:' and 'Define cast times:'. A 'Submitter' button is at the bottom of the form. The sidebar and header are consistent with the previous figure.

Figure 4.5: Create Election Page

When the election is created, the **SA** is redirected to his control panel. This has the option of going to the voter management page, question management page, trustee management page, tally management page (which is only used at the end of the election) and a button for deleting the election (see figure 4.6). On this same page, information is provided to **SA** on what he is supposed to do for the election.

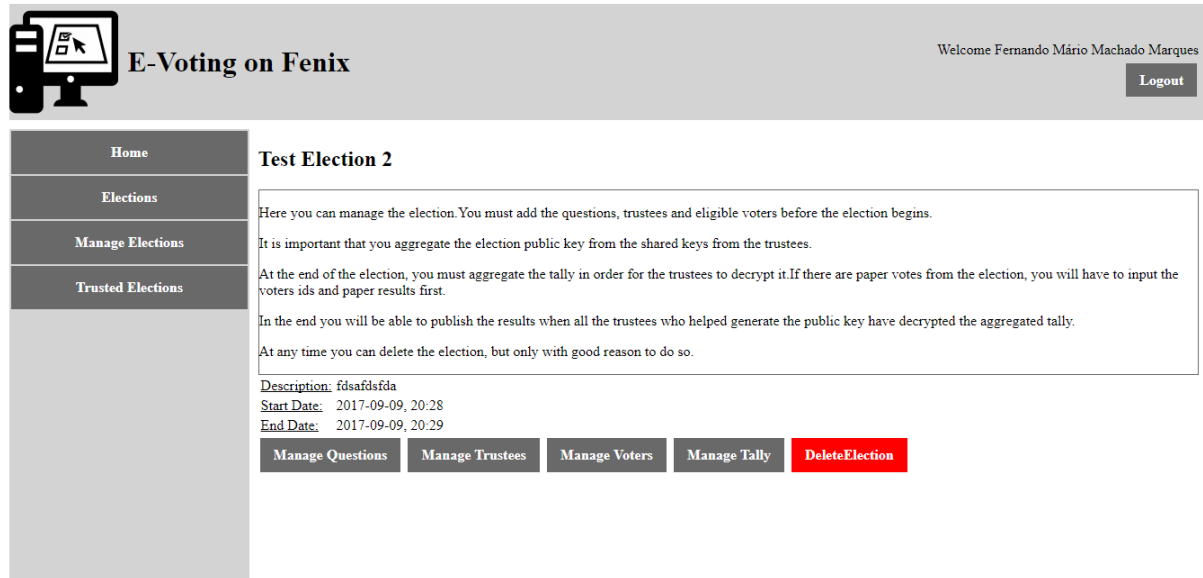


Figure 4.6: Election Administrator Control Panel

On the voter management page the **SA** will have the option of adding and removing voters from the election. To add voters, the **SA** can use the form provided to add them on by one or add them by bulk using a form for inputting .csv files with the structure present at listing 4.5. A list of all voter is available at the end of the page where an "X" is present on the left side of each voter. If the **SA** wishes to remove a voter then he only needs to press that "X". This list can be hidden and shown using a toggle button provided in order to maintain the page more organized.

Listing 4.5: Voter file format

```
1 istXXXXXX ; voterEmail@emailDomain
```

The trustee management page is very similar to the voter management page. The **SA** is only able to add trustees one by one using the form provided. A list of trustees is present at the bottom. This list also shows which trustees have participated in the key generation process. When at least one trustee has participated in that process a button appear which lets the **SA** verify the proofs and aggregate the election public key. Like in the voter management page, the trustee list has a "X" next to each trustee which allows to remove a trustee from the election.

The question management page allows the **SA** to add and remove questions and answers for the election. A question must be added one by one to the election but a question may have a variable number of answers(minimum 2). A list of questions and their respective answers is provided and like the previous pages, each question has an "X" next to them in order to remove them from the election.

The three pages mentioned above only allow management of the election before it has started. On the contrary, the tally management page only allows management after the election voting phase has

ended. If the election was defined as hybrid, then firstly the **SA** must input the paper voters and the paper voting results. Inputting the paper voters is exactly like inputting the eligible voters, with the exception that the .csv file only needs to contain a list of ids for the voters. After inputting the paper voters, each question and answer will appear for the **SA** with text inputs in order for him to submit the results for each answer of each question. After inputting the paper information (if it was hybrid, otherwise this is the page that shows), a button will appear to request the **VS** to aggregate the final electronic ballots. This page refreshes and shows which trustees have done their partial decryptions. After all the trustees have done their final decryptions, the **SA** will be able to access the tally management page where a button to aggregate and publish the results will be available.

The **SA** also has the option of deleting the election, and when chosen, a dialogue box will appear to verify that he actually wanted to do that action.

4.5.2 Trustee Pages

The trustee pages are very simple. When a trustee first accesses this page, it is to create his public key share. A button is available to do just that. After creating his key shares and sending the public part with the proof to the server, his secret key is displayed on the page for him to save. Afterwards, before the **SA** requests for the **VS** to aggregate the ballots, each time the trustee accesses this page it will inform him to wait for the end of the election. When the **VS** has aggregated the ballots, the trustees will have a button which will allow them to perform the partial decryption. After the partial decryption is done and sent to **VS**, the trustees are thanked for participating in the election.

4.5.3 Voter Pages

Last are the pages relating to the voter. The voter will most likely access the bulletin board of the election. This will provide the voter with some information regarding the election, like the dates and times, the description, the public key, the trustees and their public key shares and the eligible voters and their smart ballot trackers (see figure 4.7).

There the voter will have a vote button which allows him to access the ballot box. If the voter still doesn't have his credentials, a button will be displayed for him to register himself for the election. When he does so, his credentials will be sent to his email and the ballot box will display two text boxes to input the public and private credentials. Afterwards, and similar to what happens to the **SA**, each question and answer will appear to the voter but instead of having an input text, each answer has a radio button to select which option he wants to vote for. After all the questions are answered, the ballot is created and the voter is presented with his smart ballot tracker (which was calculated on a local script) and a option to audit the ballot or to cast it. If he chooses to cast it, then the voting process ends there. If he chooses to audit it, then a text area appears with the json of the ballot and a link to the audit page. The option to cast the ballot will also disappear. The voter must then copy his ballot, access the link for the auditing and paste it in the text area there present. Finally, when he clicks the audit button, the results will appear at the end of the page.



E-Voting on Fenix

Welcome Fernando Mário Machado Marques

Logout

Home

Elections

Manage Elections

Trusted Elections

Test Election

Description:

Test Election

Start Date:

2017-08-13, 20:30

End Date:

2017-08-25, 18:03

Public Key:

1e16dc3bcb6e49558614e6ce78fbafc49e88dece4f19fb5838edb5c00778b8c469a0585600fb21fafa75eab9fe785cf3ff6d05540cd715b3e089676b4e0fafdbf34ee3b59497ed3c4e697209afab5ff04d042f47405791f674978d1ec2943d0c2fc6274ce11f2fcb42bae6aaffbbe4b139f5bc598dadd24ad09b78d9fc486c7b

Vote


Hide Trustees Public Shares

Show Eligible Voters

Trustee Identifier	Public Key Share
ist176419	1e16dc3bcb6e49558614e6ce78fbafc49e88dece4f19fb5838edb5c00778b8c469a0585600fb21fafa75eab9fe785cf3ff6d05540cd715b3e089676b4e0fafdbf34ee3b59497ed3c4e697209afab5ff04d042f47405791f674978d1ec2943d0c2fc6274ce11f2fcb42bae6aaffbbe4b139f5bc598dadd24ad09b78d9fc486c7b

Figure 4.7: Smart Bulletin Board before publishing the results

At the end of the election, the voter can check the results present on the Smart Bulletin Board and export the election data (which is in json format) so he can verify that everything went accordingly (see figure 4.8). He will also be able to check the voter list and verify that his smart ballot tracker is there correctly. In case he voted by paper, he will verify that the smart ballot tracker is replaced with the word "paper" (see figure 4.9).



E-Voting on Fenix

Welcome Fernando Mário Machado Marques

Logout

Home

Elections

Manage Elections

Trusted Elections

Test Election 2

Description:

fdsafdsfda

Start Date:

2017-09-09, 20:28

End Date:

2017-09-09, 20:29

Public Key:

35dc4de183e8bce00d885a2f2791f79f480cecd31b50ef4ac4c9ff171a1f7546b99cb39bec1a41811a7472a8346df630654f8485d79a99227ae3a32069e18ac51e0ba1239fa76c7e61ca6f0537ea7760cdc6480cf59ca6f403b14d77e4f0d13f937c785de29ede01aec49712d5b7017323134ad6fcdb90bec1e5d056bec3b

Election Tally

Best Movie

Dark Tower 2

Death Note 1

Who is this?

Me 2

Someone Else 1

You 0

Export Election Data

Show Trustees Public Shares

Show Eligible Voters

Figure 4.8: Smart Bulletin Board after result publication

Voter Identifier	Smart Ballot Tracker
ist176419	3cc2562054923467c16c894c832c3c71d34645cbddea5906334aa8bc35d7242
ist176473	
ist176437	paper Voter

Figure 4.9: Example voter list after result publication

4.6 Specification Fulfilment

Through this section, some implementation details will be presented and compared side by side to the specification of the protocol present at section 3.3. As such this section will be divided in subsections, each representing the phases of the protocol, and each subsection will deal with the main details regarding that phase.

4.6.1 Set Up Phase

On the start of this phase the **SA** inputs the election data to **VS**. This last calculates the cryptographic parameters as shown in section 4.4.1 and saves this information in a Election object which is based on the model described in section 4.3.1 (lines 1-2 in table 3.1). A list of all the Election objects stored in the database is the equivalent of the specifications **EL**. Since the specification follows one election, let's assume that the uuid generated for this election is denoted by *electionId*. When an election is created, **CA** is also informed of the election and receives its id and the end date so that it can delete all the information regarding it 30 days after the end date defined.

It is also during this phase that the trustee can input the list of all trustees and the list of all eligible voters (lines 4 and 11 of table 3.1). Each trustee creates a Trustee object which has a connection to the Election object created earlier. The same can be said for each voter, except they are created as Voter objects. As such **TL** can be defined as the list of Trustee objects which have the foreign key for the Election object with key *electionId* and **EVL** can be defined the same way, except with Voter objects instead of Trustee objects.

The trustees perform the key share generation process during this time. Each public key share and its proof is stored on their objects, published (lines 5-6 of table 3.1) and retrieved by the **SA** in order to perform the verification of the proofs (lines 7-8 of table 3.1) and aggregation of the election public key. This public key is stored in the Election object and is also published (lines 9-10 of table 3.1).

The structures present in this phase can be described as the following sql queries in listing 4.6

Listing 4.6: Set Up Phase Structures in sql

```
1 EL = SELECT * FROM Election ;
2 TL = SELECT * FROM Trustee WHERE election=electionId ;
3 EVL = SELECT * FROM Voter WHERE election=electionId ;
```

4.6.2 Voting Phase

During this phase the voter can vote electronically, by paper or both forms. The last ballot counted is always the paper one if both are present.

If the voter votes by paper, his ids will be submitted to the system as a paper voter (line 8 from table 3.3). As such, the a query will be done in order to find his Voter object and the flag which says that he

voted by paper will be set to true. The sql equivalent, assuming istXXXXX is the identifier of the voter is display in listing 4.7

Listing 4.7: Update voter as paper voter

```
1 UPDATE VOTER SET paperVoter=True WHERE identifier := 'istXXXXX' and election=
  electionId ;
```

This creates the structure **LPV** which in sql is denoted as in listing 4.8

Listing 4.8: LPV in sql

```
1 LPV = SELECT * VOTER WHERE paperVoter=True and election=electionId ;
```

If the voter voted electronically, he must first request a credential (line 1 from table 3.2). This request is sent from **VS** to **CA** (line 2 from table 3.2) who sends both credentials to the user via email(line 5 from table 3.2) and send the public credential to **VS** which is then stored on the Voter object(lines 3-4 from table 3.2).

After receiving their credentials, the voter is able to vote. All the information needed to create the ballot is loaded before hand so that the voter is able to create it even offline. Each selected answer is chosen as value 1 and each non-selected answer as value zero. After choosing a option for every question, the ballot is encrypted, the proofs are calculated, a signature is formed and the smart ballot tracker is calculated (which the voter should keep). The ballot, which has the structure present in appendix A, is then sent to **VS** (lines 7 from table 3.2). When receiving the ballot, **VS** checks that the public credential it received is the same as the voters, checks if there is a ballot with the same public credential already present or not and checks the proofs and the signature on the ballot. If everything is correct, then the ballot is stored in a Ballot object which has a connection to the Election object(lines 8-10 from table 3.2) and **VS** creates a Smart Ballot Tracker for that ballot. A list of all the ballots objects that correspond to a specific election is the equivalent of the specifications **TEV**. The sql equivalent of this structure is present in listing 4.9

Listing 4.9: TEV in sql

```
1 TEV = SELECT * FROM Ballot WHERE election=electionId ;
```

4.6.3 Tally Phase

During the tally, the **VS** creates the **TFEV** which is a list of all electronic ballots without those of paper voters (line 1 from table 3.4). To do this, the system first gets the list of all public credentials from the Voter objects which have the paperVoter flag set to True. After getting all those public credentials, the system queries all Ballot objects and removes from that list all those whose credentials are included in the list calculated earlier. This is done each time it is needed to access this list. The sql equivalent is presented in listing 4.10

Listing 4.10: TFEV in sql

```

1 TFEV = SELECT * FROM Ballot WHERE publicCredential NOT IN (
2     SELECT publicCredential FROM Voter WHERE paperVoter = True
3 );

```

After the **SA** inputs the paper voters he will also need to input the paper results which are stored in the Election object (lines 3 and 9 from table 3.4). He also requests to the **VS** to aggregate **TFEV** which is done using additive homomorphic aggregation (line 2 from table 3.4). As such for each ballot, each answer for each question is aggregated to the equivalent of the next ballot, thus calculating the encrypted aggregated tally. This is also stored in the Election object.

Each trustee then fetches this aggregated encrypted tally and calculates the partial decryptions and proofs for these (lines 4-5 from table 3.4). These are stored on the Trustee object which the **SA** accesses in order to perform the validation of the proofs and aggregation of the partial decryptions, which gives the electronic tally (lines 5-6 from table 3.4). The electronic tally and paper tally are then added by the **SA** and sent to **VS** to be published. This are stored once more in the Election object (lines 10-12 from table 3.4).

4.6.4 Audit Phase

The audit phase is a phase that occurs in parallel with the previous phases. The main ideas to retain from this phase are:

- Voter can check their Smart Ballot Trackers in the bulletin board. One Smart Ballot Tracker is created on the front-end and the other on the **VS**. Since these are the equivalent of hashing the ballot, if both these Smart Ballot Trackers which were generated on different places are equal, then it is possible to conclude that the value that generated (the ballot) is also equal. This allows to prove that the ballot was count as cast.
- At the end of the election, everyone will be able to export the election data in order to verify that the election occurred correctly. This creates a json from the pertinent information which is stored in the objects present in the database.
- The voter can perform the audit of their ballot before casting it. Since this requires to reveal the randomness used during the encryption process, the ballot needs to be generated with another randomness to be cast again. When audited, the ballot is check to view that the answers were the ones the voter wanted, the encryption was well done, the proofs were calculated correctly and that the signature over the ballot is correct. This allows to prove that the ballot is cast as intended since the script performing the auditing is independent from the script performing the generation of the ballot.

4.7 Chapter Summary

This chapter detailed some of the technologies used for the implementation and some decisions that were done. It went into some detail regarding some of the back-end organization, like the connection between server, OAuth connection, the database organization and other. It then presented the a more close relation between the specification presented in 3.3 and the implementation.

Finally the architecture used during the development and usability tests is provided. It is also provided what changes are needed in it so that the system can be deployed in a real world scenario.

Chapter 5

Usability evaluation

In this chapter provides an analysis on the results of user testing done on the implementation mentioned on chapter 4. Firstly, a description of the tests that were done is presented. In this, the characteristics of the users, the testing environment and of the questions done to the users are presented. Afterwards the results of the evaluation are shown. These provide a basis for the discussion which is presented in the following section.

5.1 Tests Description

In order to evaluate the usability of the system, user testing was done. The servers were both deployed on the same machine running a developer server for each of the applications. Since the evaluation was run inside a closed network no secure channels were used. The user group who participated in this evaluation consisted of 15 individuals whose ages ranged from 20 years old to 55. This was so that the evaluation would received different feedback from different age groups.

The machine that ran the tests had the following specification:

CPU Intel Core i7-3517U Processor

RAM 8 GB

OS Windows 10 64-bit

An user was chosen in order to serve as a trustee for the election. This user had an explanation on how the protocol worked and what was his role. The rest of the users were informed on how the system worked in order for them to have an understanding of what was going to happen.

The election had the following questions to which the users had to answer:

Which of these movies would you like to see?

- Dark Tower
- IT
- Other

Which of these books do you like best?

- Lord of the Rings
- Moby Dick
- 1984
- None of the above

In order to simulate a real world scenario, it was informed that the votes present in table 5.1 and 5.2 would be inserted into the the system. This would simulate a scenario in which both paper and electronic voting was occurring at the same time. The total number of simulated paper voters was 12, thus making the total number of voters in the system 27.

Table 5.1: Simulated paper votes for question "Which of these movies would you like to see?"

Which of these movies would you like to see?	
Answer	Simulated Paper Votes
Dark Tower	3
It	4
Other	5

Table 5.2: Simulated Paper Votes for question "Which of these books do you like best?"

Which of these books do you like best?	
Answer	Simulated Paper Votes
Lord of the Rings	5
Moby Dick	3
1984	2
None of the above	2

As each user executed the task of voting, they would share their answers which were recorded in order to compare these with the final election results calculated by the system. In the end of the election the following questions were posed on the users:

- Did you audit a ballot?
- Did you verify your ballot was cast correctly?
- Did the system feel fluid?
- What would you improve?
- Would you use the system instead of the classical paper voting?

For the user who had the job as a trustee, it was also asked if he tough the job would be intuitive.

The results of the tests are thus described in section 5.2.

5.2 Results

The votes recorded from users are the ones displayed in table 5.3 and 5.4.

Table 5.3: Recorded Electronic Votes for question "Which of these movies would you like to see?"

Which of these movies would you like to see?	
Answer	Recorded Electronic Votes
Dark Tower	9
It	2
Other	4

Table 5.4: Recorded Electronic Votes for question "Which of these books do you like best?"

Which of these books do you like best?	
Answer	Recorded Electronic Votes
Lord of the Rings	4
Moby Dick	0
1984	3
None of the above	8

There was an user changed the answer of the first question from "Dark Tower" to "It" and the answer of the second question from "None of the above" to "Moby Dick" by voting firstly electronically and then by paper. The election tally is present in tables 5.5 and 5.6.

Table 5.5: Tally for question "Which of these movies would you like to see?"

Which of these movies would you like to see?	
Answer	Tally
Dark Tower	11
It	7
Other	9

From the list of question that were asked in the end of the election, the following results were recorded:

Did you audit a ballot? Of the 15 users who participated in the test, only 3 audited their ballots.

Did you verify your ballot was cast correctly? Of the 15 users 5 verified that their Smart Ballot Trackers were published on the Bulletin Board and were correct.

Did the system feel fluid? Most of the users said the system felt fluid although it was also said that the UI was a bit lacking.

What would you improve? The answers for this question focused mostly on these 3 main aspects:

- Improvements on the overall UI.
- Improvements on passing the ballot from the ballot box to the audit form since copy paste may lead to possible errors.
- Improvements on emphasising that it is important to save the trustee key share and the voter public credentials.
- Possibility of a credential recovery system.

Would you use the system instead of the classical paper voting? When asked this question, 9 of the users said that they would probably use the system, 3 said they would continue only to vote

Table 5.6: Tally for question "Which of these books do you like best?"

Which of these books do you like best?	
Answer	Tally
Lord of the Rings	9
Moby Dick	4
1984	5
None of the above	9

by paper and another 3 said that they would vote by paper but it would be a good feature to be implemented in case it was needed by them.

Finally to the user who also had the job of a trustee, he mentioned that the job was not difficult as long as you had a brief explanation before the first use. He also proposed a notification system so that the trustee would know when he would need to perform a task.

5.3 Discussion

This section presents a reflection on the results presented for the evaluation present in section 5.1.

Firstly, it is possible to see that the protocol is working correctly. On both questions a total number of answers done were 27, which corresponds to the 15 votes from the users who participated on the evaluation and the 12 simulated users. It is also possible to verify that the protocol takes priority on paper votes over electronic votes from the user who cast both votes. If both votes had been counted then the results present would have 12 votes for "Dark Tower" in the question "Which of these movies would you like to see?" and would have 10 votes for "None of the above" which would make a total of 28 votes for each of the questions.

From the results of the questions done after the election it is possible to verify that only 20% of the participants audited their ballots. This can be due to the current scenario being a test and the users did not feel the need to audit. As such, in a real world scenario, a larger part of the user may decide to verify their ballots. The same can be said for the verification of correct cast of the ballot. Either way, it would be a good idea to reinforce the idea that these steps are important to increase the trust of the final results of the election. One way to accomplish this is by, in the UI, presenting the importance of those optional steps to the voters. This approach can also be used to emphasise the importance of the trustee secret key share and the voter credentials.

In terms of improvements requested, the UI was a main aspect. The current UI is very minimalistic and as such in future version should contain more information for the user and be more intuitive. Important pitfalls to avoid are some that were presented by Karayumak et al. in "Usability Analysis of Helios - An Open Source Verifiable Remote Electronic Voting System" [36], for example inconsistencies in the system in regards to wording and use of buttons and links. Another problem which was mentioned in the paper and also revealed during the user test is the use of copy paste mechanism during the ballot auditing. This can be prone to error for less experienced users who, due to lack of technical skill or simple distraction, not copy the whole ballot json or even insert a foreign character which would case the

auditing to fail. One proposed solution would be to allow the user to download a file with the information and then upload it to the server for auditing, but a study would firstly need to be conducted on such approach due to the possibility of leaking information. The last improvement mentioned by the users was the possibility of credentials recovery. This is already implemented in the original Belenios protocol and should be able to be ported over to H-Belenios. One important detail to take into account would be the necessity to delete the previous credential and its association with the user id and possible previous ballot so that the relation of one to one from voter to credential is maintained.

Finally, and one of the most important questions, is whether the users would use such a system to cast a vote. 60% of the users answered that they would use such system while 20% said they would prefer only the paper voting. That leaves 20% who said they would not mind the feature since it could be useful. That gives the system a total of about 80% of approval. Since this was a small test group, these results may still vary a lot, but for now it gives a good perspective in term of the future possibility of deploying such system.

5.4 Chapter Summary

In this chapter the results of an usability evaluation were presented. It starts by describing what were the conditions in which the test took place and what were the test questions. Afterwards it presents the results of said tests. Finally, a reflection on the results of the tests is presented. In this reflection it is possible to verify that the protocol is currently functional but still has room from improvement from an usability point of view. It is also possible to verify that most users would be willing to use said protocol or even have it available in case it was ever needed. As such, it is possible to conclude that there is a possibility of implementing the system on an academic environment like Instituto Superior Técnico with a certain degree of acceptance from the users, although the current system still needs some work before it is deployed.

Chapter 6

Conclusions and Future work

This thesis presents a hybrid voting protocol that integrates an already existing e-voting protocol with the paper voting protocol which is already used. The cultural shift which is required in order to adopt a full e-voting protocol is reduced by minimizing the number of changes done on the paper voting protocol. This thesis also presents details regarding the implementation of said protocol and the results that were received from an usability test done with various users. From these results it was possible to verify that users were willing to adopt this protocol and although it still needs some improvements in some areas, it is currently functional and proves that implementing a hybrid e-voting protocol on Instituto Superior Técnico is a real possibility for the near future. It is also important to note that although the solution presented was created with the concrete case of using it in IST, and as such the implementation is coupled with the authentication mechanism provided by the university integrated campus management system (Fenix), it is possible to adapt it in order to be used in other academic environments with similar requirements. To do currently would only require to change the back-end used for the OAuth authentication in order to connect to another organizations OAuth authentication server. It would also require to change the setting were connection to the mail server is done, and some UI which would be more appropriate for other organizations.

The current solution improves on some aspects of the original two baseline protocols, like coercion resistance and a relaxed assumption on the poll workers. And even though this solution is implemented and functional, it still has some limitations and improvements that should be addressed before being deployed in a real world scenario. These are the following:

- Analysis of the security of the front-end and the back-end of the system. This is currently being carried out by two distinct thesis.
- Implement a mechanism to use Lightweight Directory Access Protocol (LDAP) in order to select voters and trustees. Currently the system relies on .csv files in order to select these users and as such is prone to user error. Another alternative, if LDAP access is not given, would be to create a .pdf file parser in order to parse the electoral rolls since currently they are released in this format.
- Implementation of a threshold shared key generation process. Currently the system doesn't pro-

vide threshold support to the shared key generation process. As such, a trustee who starts to participate in the process and then refuses to participate in the decryption can block the results from being decrypted.

- Study an alternative way to distribute credentials, for example with the use of smart cards. Currently the system uses emails in order to send the credentials to the users. This is not a secure way to distribute the credentials since emails can suffer various jumps until reaching its destination host. As such some hosts can save the email with the credentials. Even if the email is encrypted with the use of secure channels, there is no guarantee that the encryption is not broken during the time in which the election is occurring.
- Implement a centralized list of eligible voters. Currently these lists are to be kept in paper format and then, at the end of the election, to be transferred to the system by the server administrator. This is prone to user error and a centralized list would also save time after an election has ended since the poll workers could submit directly onto the system who has voted by paper in real time.
- Overall improvement on the UI since the current UI is very minimalistic and can confuse some less experienced users.
- Implementation of a notification system in order to warn the trustees when their interference with the election is necessary. Currently, no notification is given to the trustees which could lead to these forgetting their roles during the election.
- Implementation of a different system which allows for ballot auditing, in order to replace the current copy paste system, since this method is prone to human error.
- Implementation of the Belenios credential recovery mechanism. Currently no credential recovery is implemented. If a user currently loses his credentials he will not be able to vote electronically and is then forced to vote by paper if he wishes to vote.

In conclusion, this thesis provides a specification and prototype of a hybrid e-voting system to be used on an academic environment like IST. It was proven that work still needs to be done in order for it to be ready to be deployed on a real world scenario, but the results received were positive and the possibility of bringing e-voting to IST is nearer.

Bibliography

- [1] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman. Security analysis of the estonian internet voting system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 703–715. ACM, 2014.
- [2] R. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Herrnson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, et al. Scantegrity ii municipal election at takoma park: the first e2e binding governmental election with ballot privacy. In *Proceedings of the 19th USENIX Security Symposium*. USENIX Association, 2010.
- [3] B. Adida, O. De Marneffe, O. Pereira, J.-J. Quisquater, et al. Electing a university president using open-audit voting: Analysis of real-world use of helios. *EVT/WOTE*, 9(10), 2009.
- [4] M. A. Herschberg. *Secure electronic voting over the world wide web*. PhD thesis, Massachusetts Institute of Technology, 1997.
- [5] Instituto Superior Técnico. Regulamento para eleição dos órgãos do ist – normas gerais. *Diário da República*, 2.^a série — N.º 143 — 25 de julho de 2012, pages 26500–26505, 2012.
- [6] S. Glondu. Belenios specification. *INRIA technical Report*, pages 1–8, 2013.
- [7] Fernando Marques, Ana Almeida Matos, Jan Cederquist. Integrating paper-based voting and belenios – a hybrid voting protocol for an academic organisation. In *INForum 2017 Atas do Nono Simpósio de Informática*, pages 257–268, 2017.
- [8] Github project page. <https://github.com/fmarques94/E-Voting-on-Fenix>. Accessed: 16/10/2017.
- [9] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a secure voting system. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 354–368. IEEE, 2008.
- [10] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.
- [11] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [12] T. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT’91*, pages 522–526. Springer, 1991.

- [13] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
- [14] M. S. Bhigade. Secure socket layer. In *Computer Science and Information Technology Education Conference*, pages 85–90, 2002.
- [15] T. Dierks. The transport layer security (tls) protocol version 1.2. 2008.
- [16] P. Y. Ryan, D. Bismark, J. A. Heather, S. A. Schneider, and Z. Xia. The prêt à voter verifiable election system. *IEEE transactions on information forensics and security*, 4(4):662–673, 2009.
- [17] R. Oppliger. How to address the secure platform problem for remote internet voting. *Sis*, 2:153–173, 2002.
- [18] D. Chaum, R. T. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. Ryan, E. Shen, A. T. Sherman, and P. L. Vora. Scantegrity ii: End-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE transactions on information forensics and security*, 4(4):611–627, 2009.
- [19] F. Zagórski, R. T. Carback, D. Chaum, J. Clark, A. Essex, and P. L. Vora. Remoteegrity: Design and use of an end-to-end verifiable remote voting system. In *International Conference on Applied Cryptography and Network Security*, pages 441–457. Springer, 2013.
- [20] R. Joaquim, C. Ribeiro, and P. Ferreira. Veryvote: A voter verifiable code voting system. *VOTE-ID*, 5767:106–121, 2009.
- [21] C. A. Neff. Practical high certainty intent verification for encrypted votes, 2004.
- [22] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *Transactions on Emerging Telecommunications Technologies*, 8(5):481–490, 1997.
- [23] R. Joaquim, P. Ferreira, and C. Ribeiro. Eviv: An end-to-end verifiable internet voting system. *computers & security*, 32:170–191, 2013.
- [24] R. Joaquim and C. Ribeiro. An efficient and highly sound voter verification technique and its implementation. In *International Conference on E-Voting and Identity*, pages 104–121. Springer, 2011.
- [25] R. Haenni, R. Koenig, S. Fischli, and E. Dubuis. Trustvote: A proposal for a hybrid e-voting system. *Bern University of Applied Sciences, Höhweg*, 80, 2009.
- [26] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology—AUSCRYPT’92*, pages 244–251. Springer, 1993.
- [27] R. Küsters, J. Müller, E. Scapin, and T. Truderung. select: A lightweight verifiable remote voting system. In *Computer Security Foundations Symposium (CSF), 2016 IEEE 29th*, pages 341–354. IEEE, 2016.

- [28] B. Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.
- [29] K. Sako and J. Kilian. Receipt-free mix-type voting scheme. In *Advances in Cryptology—EUROCRYPT’95*, pages 393–403. Springer, 1995.
- [30] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.
- [31] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Crypto*, volume 92, pages 89–105. Springer, 1992.
- [32] V. Cortier and B. Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.
- [33] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology—CRYPTO’94*, pages 174–187. Springer, 1994.
- [34] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene. Election verifiability for helios under weaker trust assumptions. In *European Symposium on Research in Computer Security*, pages 327–344. Springer, 2014.
- [35] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
- [36] F. Karayumak, M. M. Olembo, M. Kauer, and M. Volkamer. Usability analysis of helios-an open source verifiable remote electronic voting system. *EVT/WOTE*, 11, 2011.
- [37] V. Cortier, G. Fuchsbaauer, and D. Galindo. Beleniosrf: A strongly receipt-free electronic voting scheme. *IACR Cryptology ePrint Archive*, 2015:629, 2015.
- [38] Java. <https://www.java.com>. Accessed: 16/10/2017.
- [39] Python. <https://www.python.org/>, . Accessed: 16/10/2017.
- [40] Node.js. <https://nodejs.org/en/>. Accessed: 16/10/2017.
- [41] Django web framework. <https://www.djangoproject.com/>. Accessed: 14/10/2017.
- [42] Python social auth. <https://python-social-auth.readthedocs.io/en/latest/>, . Accessed: 14/10/2017.
- [43] Oauth. <https://oauth.net/>. Accessed: 16/10/2017.
- [44] Tom wu stanford page. <http://www-cs-students.stanford.edu/~tjw/jsbn/>. Accessed: 16/10/2017.

- [45] vkthread page. <http://www.eslinstructor.net/vkthread/>. Accessed: 16/10/2017.
- [46] Stanford javascript crypto library page. <http://bitwiseshiftleft.github.io/sjcl/>. Accessed: 16/10/2017.
- [47] Web crypto api. https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API. Accessed: 16/10/2017.
- [48] P. Gaudry. Some zk security proofs for belenios. *INRIA technical Report*, 2017.

Appendix A

Ballot Example

Listing A.1: Example of ballot

```
1 {
2     '503d45d0-fc5d-4423-a089-dba68b9099d4': {
3         'answers': {
4             '1142d97e-0763-4f9d-9ec5-8960c081bdb7': {
5                 'alpha': '',
6                 'beta': '',
7                 'individualProof': [{
8                     'challenge': '',
9                     'A': '',
10                    'B': '',
11                    'response': ''
12                }, {
13                    'challenge': '',
14                    'A': '',
15                    'B': '',
16                    'response': ''
17                }
18            ]
19        },
20        'fff72368-17c1-430c-9264-412b37416181': {
21            'alpha': '',
22            'beta': '',
23            'individualProof': [{
24                'challenge': '',
25                'A': '',
26                'B': '',
```

```

27         'response': ''
28     }, {
29         'challenge': '',
30         'A': '',
31         'B': '',
32         'response': ''
33     }
34 ]
35 },
36 'ce707d5e-b425-423a-b108-12a42184f93a': {
37     'alpha': '',
38     'beta': '',
39     'individualProof': [{
40         'challenge': '',
41         'A': '',
42         'B': '',
43         'response': ''
44     }, {
45         'challenge': '',
46         'A': '',
47         'B': '',
48         'response': ''
49     }
50 ]
51 }
52 },
53 'overall_proof': [{
54     'challenge': '',
55     'A': '',
56     'B': '',
57     'response': ''
58 }, {
59     'challenge': '',
60     'A': '',
61     'B': '',
62     'response': ''
63 }
64 ]
65 },

```

```
66     'signature': ['', '']
67 }
```

