FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Electronic Voting over the Internet – A real-world solution

Michael Seufert

WORKING VERSION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Dr. Manuel Barbosa

Second Supervisor: Renato Portela

January 27, 2017

Resumo

Sistemas electrónicos de votação ainda não conseguiram impor-se como alternativa à tradicional eleição por papel, quer em eleições privadas quer em eleições oficiais. Ainda assim, nos últimos anos, o trabalho científico na área produziu ferramentas e soluções que tornam estes sistemas mais fiáveis, confiáveis e verificáveis. Esta evolução foi acompanhada com o desenvolvimento de soluções comerciais pela indústria.

Uma solução desenvolvida em Portugal e utilizada em eleições particulares é um sistema de votação pela Internet chamado Certvote, desenvolvido pela Multicert - uma grande companhia portuguesa na área da segurança digital. A Multicert abordou a Universidade do Porto com o desejo de ver o seu sistema analisado em termos de como se posiciona no estado da arte em segurança de sistemas, criptografia e em geral no campo da votação electrónica.

Este trabalho propõe-se a estudar o estado da arte para determinar que propriedades são esperadas de sistemas de votação electrónica e que ferramentas criptográficas tipicamente são utilizadas para alcançar essas propriedades. Procedemos depois para uma descrição e análise do sistema Certvote, determinando como se relaciona com as propriedades definidas. Na procura duma referência, analisou-se também um sistema chamado Helios, sistema de votação bem estudado com propriedades de verificabilidade extremo a extremo.

Por fim procedemos à sugestão e desenvolvimento de algumas alterações ao sistema Certvote, mostrando que essas alterações permitem ao sistema mimetizar muitas das propriedades de Helios, colocando-se assim num patamar superior em termos de propriedades de segurança.

Abstract

Electronic voting systems have yet to break through as alternatives to traditional paper voting, both in private as in official elections. Still in the last years scientific research in this field has produced tools and solutions that make these systems more reliable, trustworthy and verifiable. This evolution went side-by-side with the development of commercial solutions by the industry.

One solution that is developed in Portugal and has been deployed for private use is an internet voting system called Certvote, by Multicert - a major Portuguese company in the field of digital security. Multicert approached the University of Porto with the wish to have its system analyzed in terms of how it positions itself with the state of the art in system security, cryptography and the general field of electronic voting.

This work aims at studying the state of the art to determine what properties are expected from electronic voting systems and what cryptographic tools are generally used to achieve said properties. We then proceed to describe and analyze Certvote, determining how it relates to the defined properties. In order to have a reference, we also analyzed Helios, a well-studied end-to-end verifiable voting system.

Finally we proceed to suggest and develop improvements to Certvote, while showing how these improvements allow Certvote to mimic a lot of the properties of Helios, enhancing its security properties.

Acknowledgements and Dedication

I would like to thoroughly thank everyone who helped in the development of this work. Primarily, of course, professor Manuel Barbosa and Renato Portela from Multicert, who both made my work possible. It was an absolute pleasure to work with both.

Also this work would not have come to be without the people at Multicert who took their time to discuss with me all the intricacies of the system they develop. I have to refer Pedro Cunha (who was also always so quick answering my e-mails, thanks a lot!) and Tiago Teixeira who sat down with me every time I needed - but in fact I ended up having the pleasure of interacting with a whole lot of people. Thank you all!

The DCC at Faculdade de Ciências da Universidade do Porto offered me a convenient place to work, and professor Barbosa's PHD students on-site were always a source of valuable inputs.

The Faculdade de Engenharia da Universidade do Porto was a wonderful school - truly a school of life, as we say in Portuguese - to guide me through a rather long, interrupted, journey. I was inspired by a lot of colleagues and teachers but I have to point out professors Adriano Carvalho and Carlos Costa with whom I served on the Academic Affairs Council and on the Board and from whom I learned so much more than just engineering. My life outside FEUP would have been very different had I never had the chance to discuss with them so many aspects of the workings of an institution like FEUP and a program like LEEC/MIEEC.

But it was my family, namely Susana and Francisco but also my mother and my brother, who paid the toll for the time invested in this journey and lost to them. To them all, to my late and dearly missed father (every day, Papa) and to the little being (it's a girl!) who started growing midway into the development of this dissertation, I humbly dedicate my work.

Michael Seufert

"That to secure these rights, Governments are instituted among Men, deriving their just powers from the consent of the governed"

Declaration of Independence

Contents

List of Figures

Abbreviations

AES	Advanced Encryption Standard
BPS	Ballot Preparation System
CBC	Cipher Block Chaining
CPA	Chosen Plaintext Attack
CSV	Comma-Separated Values
E2E	End-to-End
HSM	Hardware Security Module
IV	Initialization Vector
NIST	National Institute of Standards and Technology
OAEP	Optimal Asymmetric Encryption Padding
PBB	Public Bulletin Board
RSA	Rivest-Shamir-Adleman cryptosystem
SMV	Sistema de Mediação de Voto
SOTA	State of the Art
SSL	Secure Sockets Layer
TLS	Transport Layer Security
VC	Voting Commission
WWW	*World Wide Web*
ZKP	Zero-Knowledge Proofs

Chapter 1

Introduction

1.1 Context

Voting is the fundamental act in Democracies but also in a lot of organizations. Most people will probably cast their first vote at school, when electing a class representative. Schoolmates will stand as candidates and papers will be filled with names to be folded and counted by the teacher who most probably keeps track of the votes on the blackboard while counting.

This of course is but the first moment one has to cast a ballot - later maybe a student union has to be elected and in university students will help to elect the Dean. Reaching voting age finally opens the gates to the "big thing": voting for Members of Parliament, the Mayor or the President makes us fell part of our community.

We expect elections to be run in a way that results are honest, meaning the ballot we placed in the urn will be counted and added to the total after polls close. We also expect the vote to be kept private so even in the days of Social Media and Real-Time Streaming from mobile phones we know that it is up to us to speak about our choice or to keep it undisclosed as it is our right.

On the other hand, of course, if votes are private and urns impenetrable to the peering eye then we need to trust the system and polling staff that our vote indeed is counted. Privacy goes in hand with opaqueness: a hidden vote is hidden and we cannot trace it later, when results are announced.

What may strike us as odd, is that in the age of touchscreens and 24/7 Internet connectivity we still are voting with the good old paper ballot. Electronic voting systems have been discussed often enough and in organizations they are used, in Portugal, for over ten years now[1] but usage remains low and since a pilot project in 2004/2005[2] on a national scale, no steps have been taken towards implementing an electronic voting system for official elections.

One of the system providers that participated in the pilot project was Multicert, a Portuguese company that operates in the field of digital security. Notably, Multicert operates the digital certificates on the Portuguese ID-cards and is a root certificate authority. In addition to that they also

[1] http://tek.sapo.pt/noticias/internet/artigo/bancarios_aderem_a_sistema_de_voto_electronico_para_eleicoes_do_sindicato-878515tek.html, accessed January 2017

[2] http://www.umic.pt/index.php?option=com_content&task=view&id=3047&Itemid=418, accessed January 2017

develop Certvote, an electronic voting solution, which was used in the Portuguese pilot project.

Ten years passed since that project, Multicert approached the University of Porto in order to obtain an academic view of their implementation namely to understand how it fits itself in he State of the Art of research in the field of electronic voting. While not necessarily aiming at the participation in official elections, that future possibility was put on the table. Appropriately, news made public on the last days of this work mentioned the intention of the Portuguese government to work out an electronic voting scheme for Portuguese citizens living abroad[3].

In an age where election hacking is making headlines[4] such an analysis is an interesting challenge.

We will try to define what properties an electronic voting system must show, as per the scientific literature. Our goal will be to see how these properties can be achieved and then analyze Certvote from an academic point of view and try to fit it into that set of properties. Furthermore we will suggest changes that enable Certvote to fulfill more of those properties while developing a JAVA proof of concept for one of those changes.

We will not want to make a point for or against the use of electronic voting in official elections, though. That is a discussion that goes far beyond the cryptographic properties and workings of a particular electronic voting system although these will surely be an essential point in such a discussion. Works like the present one may pave some of the way in that discussion.

1.2 Goals

Our main goals, defined initially when this work started, were the following:

1. Investigation of the state of the art for electronic voting systems in the scientific literature;
2. Detailed characterization of Certvote with the aid of Multicert's development team;
3. Comparison of Certvote and relevant alternative solutions both in terms of specific scenarios it should work under and of security requirements or trust models it offers;
4. Proposition of changes to improve Certvote according to the obtained results.

1.3 Methodology

The work that originated this text, was roughly done between the start of September 2016 and the end of January 2017. While the in-house contact with Multicert was essential, we started to read about cryptographic voting before first contacting with the development team of Certvote. Our reasoning was to try to see how generally the issue of e-voting is approached in academic works before looking at the solution we wanted to analyze.

[3] http://observador.pt/2017/01/24/governo-esta-a-trabalhar-em-solucoes-concretas-para-voto-eletronico accessed January 2017

[4] The US presidential elections 2016 happened right in the middle of our work

After a first look at the State of the Art (SOTA) we were able to do a kick-off with the people at Multicert, where we presented our rough SOTA findings also laying out a set of properties for electronic voting systems. This presentation is included as Appendix A and enabled everyone with whom we were working with to be on the same page as of requirements, properties and usual approaches when developing an electronic voting protocol. We then started to meet with designated people at Multicert, at their office in Porto, to learn about the working of Certvote. We focused on the cryptographic and security properties, which were to be centrepiece of our work, while leaving aside issues like usability or protection against denial-of-service attacks. These, of course, are also essential for a real-time voting system that wishes to serve a broad population but are outside the scope of our work. Some of the information presented by the developers at Multicert can be found in the Appendixes.

With that information we went back into research to try and position Certvote in the framework of properties we defined. The analysis of Helios - an end-to-end verifiable i-voting system - helped to find out how systems that are deployed in the real world solve issues like the need for privacy and the possibility to verify if voting systems are actually encrypting votes correctly. The workings of Helios then inspired us when searching for improvements for Certvote. We wanted to keep our suggestions in a way that allowed them to be deployed inside the present system without major changes.

Our suggestions lead to improvements that allow Certvote to need less trust to be considered safe, as important steps in the handling of the votes can be made verifiable.

In the end we were also able to program a Java application that proved the concept of vote auditing.

1.4 Structure

Besides this Introduction, the following chapters comprise an analysis on the 2 "State Of The Art" where we will look at what properties can define an electronic voting system and what cryptographic primitives can be used to pursue said properties. In chapter 3 "Certvote" we will present Certvote, describing its operation, and analyzing it vis-a-vis the properties defined before.

Chapter 4 "Helios" will present an internet voting system called Helios that is well researched into by the scientific cryptographic community and we will want to understand how it implements cryptographic schemes to achieve a correct, private and verifiable voting system. This analysis takes us into chapter 5 "Suggestions on Certvote Development" where we will reanalyze Certvote with the learnings from Helios in our mind so we can make suggestions on how to improve Certvote to achieve a better fulfillment of the properties under consideration.

Chapter 2

State Of The Art

In this chapter we review what the scientific literature has produced in the field of electronic voting, focusing on solutions that work over the Internet.

2.1 Introduction

The idea to unite cryptographic tools and voting is probably as old as cryptography. In fact the very act of folding a paper, to hide its contents, and deposit it in an urn to be revealed only at a given time is in a way reminiscent of several applications of cryptography. It is thus only natural that with the advent of modern cryptography, electronic and digital tools have been developed that found their way into electronic voting protocols. This indeed happens for at least 35 years.

In 1981 David Chaum [2] proposed what essentially would later evolve into mix-nets, protocols we will describe later. Although his main aim was to provide secure anonymous communication channels, one projected use Chaum foresaw was in «[e]lections in which any interested party can verify that the ballots have been properly counted». These, he postulated, «are possible if anonymously mailed ballots are signed with pseudonyms from a roster of registered voters» - his key contribution at the time (he would go on doing many more) being the anonymous mailing protocol. The first version of Helios, in 2008 [3], in fact used mix-nets as part of the obfuscation between vote and voter.

In 1992 and building on, amongst others, David Chaum's work, Fujioka et.al. came up with a voting protocol (commonly referred to as FOO for the initials its authors) that was built on a complete notion on how to define a secure voting system[4]. This definition stills reads remarkably well:

> «**Completeness**: All valid votes are counted correctly.
>
> **Soundness**: The dishonest voter cannot disrupt the voting.
>
> **Privacy**: All votes must be secret
>
> **Unreusability**: No voter can vote twice.
>
> **Eligibility**: No one who isn't allowed to vote can vote.

Fairness: Nothing must affect the voting. [(e.g., preliminary results are leaked)]

Verifiability: No one can falsify the result of the voting»

More recent works [5, 6, 7] have also named the desired characteristics of an e-voting system and have taken it up to present the main cryptographic tools used by such systems as well as listing systems that have been introduced in the literature or actually used in real-world applications.

We will start to look at what these characteristics may be and will then follow Bernhard and Warinshi [5] and introduce Blinded Signatures, Mix-nets, Homomorphic Encryption, Threshold Encryption as well as ElGamal encryption, which is a cryptographic system that has several properties that can be used to accomplish an electronic voting protocol - as we will then see in Chapter 4.

2.2 Voting

Looking back at the definitions given by [4], we could start to note that they are quite universal - they apply both to paper and electronic voting. When discussing voting systems, what differs is how a system tries to guarantee each of these definitions. Paper and electronic systems go different paths and of course, since 1992, a lot has been written on the subject of how to deal with electronic elections.

Looking for less academic and more institutional ways to characterize how proper elections should look like, we find that the Council of Europe has produced a detailed recommendation[1] on the issue of e-voting. De Vries and Bokslag in their paper "Evaluating e-voting: theory and practice" [6] refer to it to produce the following requirements for a democratic e-voting solution:

«**Universal suffrage**: all human beings have the right to vote and to stand for election, subject to certain conditions such as age and nationality.

Equal suffrage: each voter has the same number of votes.

Free suffrage: the voter has the right to form and to express his or her opinion in a free manner, without any coercion or undue influence.

Secret suffrage: the voter has the right to vote secretly as an individual, and the state has the duty to protect that right.

Direct suffrage: the ballots cast by the voters directly determine the person(s) elected.»

These principles are reminiscent of article 21 of the Universal Declaration of Human Rights: «The will of the people shall be the basis of the authority of government; this will shall be expressed in periodic and genuine elections which shall be by universal and equal suffrage and shall be held by secret vote or by equivalent free voting procedures»[2], and are "translated" by De Vries and Bokslag into the following systemic requirements:

[1] Recommendation Rec(2004)11 adopted by the Committee of Ministers of the Council of Europe on 30 September 2004

[2] http://www.un.org/en/universal-declaration-human-rights/, November 2016

> «**Transparency/Integrity**. To make sure that the general public, as well as other key stakeholders in the electoral process, have confidence in the e-voting solution.
>
> **Ballot secrecy/privacy**. To protect the secrecy of the vote at all stages of the voting process.
>
> **Uniqueness**. To ensure that every vote cast is counted and that each vote is counted only once.
>
> **Voter eligibility**. To ensure that only persons with the right to vote are able to cast a vote.
>
> **Verifiability/auditing**. Ideally, the voter can check if his vote is counted in the end result. If not, independent auditors should be able to check the integrity of the election result.
>
> **Accessibility**. To guarantee accessibility to as many voters as possible, especially with regard to persons with disabilities.
>
> **Vote freedom/coercion resistance**. To maintain the voters right to form and to express his or her opinion in a free manner, without any coercion or undue influence.
>
> **Availability**. To ensure that the voting solution is available to the voter during the election.»

We should notice these, too, apply equally to paper and electronic voting, in fact the article we have just quoted proceeds to check how paper voting fulfills these requirements. Notably absent, though, when compared with Fujioka et.al., is the notion of Fairness, understood as immunity against preliminary leakage of results. Apart from that, the notions of Availability, Coercion Resistance, Accessibility and Transparency are new and should not be considered minor, but we will want to take a closer look at what Privacy and Verifiability mean.

Ali and Murray[7] gave some attention to these two properties, and expanded them both. According to them, a private system should never reveal how the voter casts his vote (**Ballot Secrecy**), should never give the voter any evidence of how he voted (**Receipt-Freeness**[3]) and should allow the voter to vote according to his choices even in presence of a coercer (**Coercion-Resistance**).

In terms of Verifiability they distinguish between Individual Verifiability and Universal Verifiability. The first property means «a voter can verify that her vote is included in the set of all cast votes», the second one that «an observer can verify that the tally has been correctly computed from the set of all cast votes».

Finally if a system is to be end-to-end verifiable, then, still according to Ali and Murray, it must also present the possibility to verify that:

- A vote was **Cast-as-intended**, meaning «the voter can verify the voting system correctly marked her candidate choice on the ballot.»

[3] Systems giving out receipts that **allow to deduce the contents of the vote** enable coercers to demand proof from a coerced voter. This should never be possible.

- A vote was **Recorded-as-cast**: «the voter can verify that her vote was correctly recorded by the voting system».
- A vote was **Tallied-as-recorded**: «the voter can verify that her vote was counted as recorded».

As they note, if a voter can verify his vote was cast-as-intended and recorded-as-cast then the system offers individual verifiability. If it is further possible for an observer (or a voter, for that matter) to verify the votes were tallied-as-recorded, then system is also universally verifiable. And these now, we may add, are properties that are only achievable with electronic voting protocols. While in paper voting, cast-as-intended verifiability is given by default (because voters know for sure what their folded ballot contains) recorded-as-cast and tallied-as-cast guarantees are not achievable. A voter must trust the people who handle the urn and the ones who count, to believe his vote was recorded and counted as cast. This is thus a major difference between paper and electronic voting and the trust each of these systems imply.

We will try to sum up the essential properties an electronic voting system should present as follows:

- **Correctness**
  - **Completeness** - all votes are counted.
  - **Soundness** - bad behaviour does not corrupt the process.
- **Eligibility** - only those allowed to vote can vote.
- **Fairness** - results are secret until the polls close.
- **Privacy** - the vote is secret, no receipt is given out linking a voter to how he voted and voters are generally free from coercion.
- **Verifiability** - voters (Individual Verifiability) and observers (Universal Verifiability) can verify, to a certain degree, the completeness of the result, ideally end-to-end.
- **Uniqueness** - every voter only votes once and every vote is only counted once.

Definition 2.1: Essential e-voting properties

We will not focus on **Usability**, **Accessibility** and **Transparency** in this work, as our goal is to discuss voting systems from a cryptographic point-of-view. They are referred for completeness sake and should generally not be forgotten when discussing voting systems.
This definition leaves us ready to take a look at how paper voting works and how it fits into the framework of properties we just presented.

2.2.1 Paper Voting

Paper voting in Portugal works roughly in the following way:

- There is a national voter register. Every voter has a voter's number he is assigned to at age 17[4] and which may change with residency. This voting number is in turn assigned to a polling station when the citizen turns 18. Information on his number and polling station can be obtained for example via SMS or the Internet before or during voting day[5].

- On election day the voter gets to his polling station and presents his ID card and voter's number. This will enable polling staff - appointed by the candidates in every election[6] - to confirm the voter is registered at that location and has not yet voted. He receives the ballot paper(s), uses a voting cabin and deposits a folded ballot paper in an urn whereupon his citizen-card is returned.

- Early vote under certain circumstances[7] is allowed as well as postal voting for citizens registered abroad[8]. Proxy vote, e.g., someone voting in the name of someone else, is not allowed.

- When polls close every urn is counted by the staff attending to it, results are centrally collected and published.

Now we can take a look at this voting scheme from the perspective of Definition 2.1. This may strike as odd, as we wrote Definition 2.1 up as *electronic* voting properties, but if we want to evaluate the differences between paper voting and electronic voting then it can be useful to see how paper voting manages to achieve properties we are looking for in electronic voting systems.

- **Correctness**
  - **Completeness** - Implies trust in the voting officials
  - **Soundness** - Short of stealing an urn, soundness seems guaranteed. Namely, manipulating a ballot paper should yield the vote null without more trouble for the tallying.

- **Eligibility** - Implies trust in voting officials. Corrupt officials could theoretically let an unauthorized person vote, marking an authorized person who never showed as present instead.

- **Fairness** - Seems guaranteed unless corrupt officials open an urn before time and publicize results.

- **Privacy** - Guaranteed if everything is normal at the polling station - a hidden camera would be a problem for ballot secrecy, though. Coercion can happen if voting postally or if a coercer can demand that a voter takes a picture of his vote.

[4] http://www.cne.pt/faq2/95/3
[5] http://www.cne.pt/faq2/108/3
[6] http://www.cne.pt/faq2/104/3
[7] http://www.cne.pt/faq2/106/3
[8] http://www.cne.pt/faq2/113/3

- **Verifiability** - recounts can happen after the election, but corrupt officials could manipulate votes in the urn - usually representatives of different candidates will handle a given urn, though.
- **Uniqueness** - Copies of official ballot papers may be smuggled in by voters and cast as regular votes, in small amounts.

This analysis is consistent with De Vries and Bokslag [6] although they ignore the possibility of coercion in a traditional voting setting with a closed urn. To us it seems that with the ubiquitousness of small cameras (incorporated in mobile phones, for instance), a coercer can nowadays expect to get a picture of a filled ballot paper. This significantly raises the possibility of coercion in paper voting to levels that were not feasible just a few years ago.
In general we can say that if voting officials work honestly then paper voting, as defined, is safe and trustworthy. Namely Correctness is, in this case, guaranteed. As voting officials are appointed by parties with conflicting aims, there is low probability of collusion.
Privacy in the election booth may be breached if (hidden) cameras are installed but such an occurrence is unheard of. Equally coercion may indeed happen as we have already referred to.

2.2.2 Electronic Voting

Our work is about Certvote, a software that sets up elections so that voters can vote from their homes. So we will not be speaking about voting machines or about presential voting at a polling station or anything similar. We want to focus on voting solutions that can be used from any place, ideally from any internet-connected computer, via a web browser or a dedicated application. This in fact means we will be concentrate on Internet voting, or **i-voting**, which is a subgroup of the electronic voting systems that exist. For now, rather than describe a given i-voting system we will look at the cryptographic primitives upon which most of the i-voting systems in the literature are built on and will try to show what kind of issues they try to tackle. Throughout this work we may use the terms electronic voting (e-voting) and internet voting (i-voting) interchangeably, while referring to voting over the Internet.

2.3 Primitives

As stated before we will follow Bernhard and Warinshi [5] in their approach and for now we assume digital signatures and public-key cryptography are well-known and need no further detailing before being introduced in section 2.5.

2.3.1 On Structure and Notation

In this section, we will consider some hypothetical election schemes to clarify why we are considering the primitives that are discussed. Each presented election setting will present obvious problems with respect to the properties in 2.1,which the primitives we discuss can solve.

As for the notation, we will consider that when using an encryption key k_e and an encryption algorithm to encrypt a message m, we will obtain a ciphertext c. Using the key k_d and c with a corresponding decryption algorithm we will obtain the original message m.
This is symbolized as follows:

$$\text{Encrypt}(k_e, m) = c \tag{2.1}$$

$$\text{Decrypt}(k_d, c) = m \tag{2.2}$$

Note that sometimes k_e and k_d might be the same key.
When we speak of a ballot in this section, we are considering it consists of a digital list of identifiers each associated to a different ciphertext. Each identifier represents a candidate or a question on the ballot, and the ciphertext represents the box in front of the candidate/question, as on a paper ballot. The voting system should encrypt a '1' or '0', respectively, if the voter chooses to tick or not to tick a box. The counting can then be achieved by adding all the '1's and '0's the voters encrypted, thus giving the total tally for each candidate/question on the ballot.
So a ballot B is defined as:

$$B = [(id_1, option_1), ..., (id_i, option_i), ..., (id_n, option_n)]; \qquad option_i \in 0, 1 \tag{2.3}$$

Definition 2.2: Ballot

An encrypted ballot, eB, looks as follows:

$$\begin{aligned} eB(key, B) =& \text{Encrypt}(key, B) \\ =& [(id_1, \text{Encrypt}[key, option_1]), ..., \\ & (id_i, \text{Encrypt}[key, option_i]), ..., (id_n, \text{Encrypt}[key, option_n])]; \qquad option_i \in 0, 1 \end{aligned} \tag{2.4}$$

Definition 2.3: Encrypted Ballot

As Bernhard and Warinshi noted[5], such a ballot format is not possible for all kinds of elections: write-in candidates, for instance, are not possible[9]. But it will be sufficiently flexible for us to work with. The sub-section on Mix-nets presents a solution that can work with any kind of ballot format. We will assume that in election schemes, ballot syntax is public.

[9] Nor are they allowed in Portuguese official elections

2.3.2 Blinded Signatures

If we start by trying to create a simple election protocol, then a simple one could work the following way: have a voter construct his ballot and send it to an authority that holds the vote register to have it signed. Then, take the signed vote and have it sent to a counting system using an anonymous channel. As the signature guarantees the voter is entitled to vote, if we presume the register is honest and additionally does not sign a vote twice for the same voter, the counting authority can add the vote to the roster and count it whenever it is supposed, without knowing who the voter is in particular as he used an anonymous channel.
One flaw with such a system is that the voter register signs a vote in the clear and thus privacy is in fact not guaranteed as we are merely shifting trust from one authority to another. It would be useful to have a way to sign a message without knowing its contents - this is where blind signatures come into play.
A blind signature scheme allows a voter to blind his vote before sending it to a voting authority, sending a blinded vote instead - the blinded vote looks like a random string and does not allow an attacker to guess the content of the vote. The authority can sign this blinded vote and send blind signature and blind vote back to the voter. He then can extract his original vote and a signature on this vote and send it to the counter who gets an authenticated vote.
To be secure, such a scheme must work in a way that no one can forge a signature that was not blind signed and it must not allow an attacker to learn the contents of the message when seeing the blinded message. For our work we will not need to detail any further how Blinded Signatures work. We can refer to Bernhard and Warinshi[5] for a detailed description and to see how to use RSA for this to build such a scheme.
Let it finally be clear that the issue of revealing the contents of the vote is but one problem with our simple electronic voting protocol. For instance, without giving up privacy, a voter never could be sure that his vote was indeed counted.

2.3.3 Homomorphic Encryption

What would make an electronic election verifiable to some degree, is if voters would see their votes published after they sent them in. Of course doing this with plaintext votes would sacrifice privacy and is not an option. But if the counting authority publishes a public key, every voter could encrypt his vote with this key, and send in a ciphertext (and a blind signature on the ciphertext, for authentication). The authority could then publish the ciphertext and everyone could check if their vote was effectively *recorded-as-cast* - and without the secret key no one could know more about the votes until the authority decrypts and starts counting. This still has problems, though. A dishonest authority might never publish the results if she dislikes them. Also, decrypting individual votes is time consuming.
Ideally we would never need that any authority decrypts individual votes. This guarantees privacy even if authorities can't be trusted, is computationally faster and precludes the need to trust anyone

with secret keys. And indeed, using homomorphic encryption, it is possible to build a voting system where no vote is individually decrypted.
Homomorphic Encryption schemes allow us to perform operations on ciphertexts that result in a new ciphertext which, when decrypted, corresponds to the result of another operation performed on the original plaintexts.
Quoting Bernhard and Warinshi[5][10]:

> *An asymmetric encryption scheme* $E = (\text{KeyGen}; \text{Encrypt}; \text{Decrypt})$ *is homomorphic if there are these additional operations:*
>
> *-An operation* $+$ *on the message space.*
>
> *-An algorithm* Add *that takes a public key pk and two ciphertexts* c_1,c_2 *and outputs another ciphertext s.*
>
> *The correctness condition is that for any messages* m_1,m_2 *the following returns* $d = m_1 + m_2$*:*

$$
\begin{aligned}
(p_k, s_k) &\leftarrow \text{KeyGen}();\\
c_1 &\leftarrow \text{Encrypt}(p_k, m_1);\\
c_2 &\leftarrow \text{Encrypt}(p_k, m_2);\\
c &\leftarrow \text{Add}(p_k, c_1, c_2);\\
d &\leftarrow \text{Decrypt}(s_k, c);
\end{aligned}
$$

This means with the proper encryption scheme and ballot formatting we can construct a voting system that uses the homomorphic property to tally the votes by performing an operation on all the encrypted votes. The result should be a new ciphertext that corresponds to the **sum** of all individual messages: If the ballot is constructed in such a way that only '0' or '1' is encrypted for every candidate (or question in a yes/no referendum), the final sum would give us the total number of votes for each option. Thus, with only one decryption we could tally a result for a large number of votes without decrypting the votes individually.
We will later see how ElGamal allows just that and we will also address how it is possible to avoid that a dishonest voter encrypts a '2',e.g., to gain more votes for a candidate in the final sum. But right now we still face, at least, one problem: with homomorphic encryption a secret key still exists and is held by an authority that could decrypt individual votes. This could mean one could learn or leak early results, harming the **fairness** of the election, and possibly link individual votes to voters, breaking **privacy**.

2.3.4 Threshold Encryption

As we just saw, we still face one problem if we want to construct a voting protocol using cryptographic tools: when using public-key cryptography, leaving the secret key in the hands of one

[10] It is their Definition 13

authority allows this authority to leak results early (breaking **fairness**) or to decrypt individual votes (harming **privacy**). What we thus need is to take this power away from a single authority.
An encryption scheme that allows threshold encryption will have authorities generate n public and secret key shares. The public key shares, pk_i, can then be combined into a public key, pk that will be used to encrypt messages as in a regular public-key environment.
To decrypt, each holder of a secret key share sk_i uses it to produce a decryption share, d_i. k shares are then needed to produce a plaintext out of the ciphertext.
We have just described a so called "k-out-of-n threshold scheme". It requires that at least k authorities out of n join to decrypt a vote - or indeed the homomorphic sum of votes. A secret key is never generated so unless you have k dishonest authorities no individual vote can ever be decrypted and there is no way you can leak results before the proper time as $k-1$ shares should provide no information.
For a more formal definition we will again turn to Bernhard and Warinshi[5][11]:

> *A (k,n) threshold encryption scheme consists of a key generation protocol* KeyGen *for n authorities and four algorithms*
>
> $$(\text{CombineKey}, \text{Encrypt}, \text{DecryptShare}, \text{Combine})$$
>
> *The key generation protocol results in all participants obtaining a public key share pk_i and a secret key share sk_i. The key combination algorithm takes a list of n public key shares and returns a public key $pk \leftarrow \text{CombineKey}(pk_1, ..., pk_n)$ or the special symbol $\perp$ to indicate invalid shares. The encryption algorithm works just like non-threshold encryption: $c \leftarrow \text{Encrypt}(pk, m)$. The decryption share algorithm takes a secret key share sk_i and a ciphertext c and outputs a decryption share $d_i \leftarrow \text{DecryptShare}(sk_i, c)$. The recombination algorithm takes a ciphertext c, a set $D = \{d_i\}_{i \in I}$ of at least k decryption shares and outputs either a message m or the symbol $\perp$ to indicate failure.*
>
> *The correctness condition is that for any message m and any set I of at least k authorities, the following yields $d = m$:*
>
> $$((pk_1, ..., pk_n); (sk_1, ..., sk_n)) \leftarrow \text{KeyGen}();$$
> $$pk \leftarrow \text{CombineKey}(pk_1, ..., pk_n); c \leftarrow \text{Encrypt}(p_k, m);$$
> $$\text{for } i \in I : d_i \leftarrow \text{DecryptShare}(sk_i, c); d \leftarrow \text{Combine}(c, \{d_i\}_{i \in I});$$

2.3.5 Mix-nets

If we want to have an election scheme that can encrypt any type of vote (like write-in votes, for instance, and not just 'yes/no' or '1/0' ballots) and still is verifiable, we will not be able to use the homomorphic property to help us count the vote. What will be of use are mix-nets. They were,

[11] Definition 19 in their paper

as stated before, first proposed by Chaum [2] for use in anonymous communication or indeed in voting protocols.
Mix-nets allow us to hide the relationship between a given ciphertext and the originator of the underlying message. What they essentially do is mix the order of a given list of ciphertexts while performing an operation on the ciphertexts themselves, in one of two ways[8]:

> *«In Chaumian mix-nets, each input ciphertext is calculated by encrypting the plaintext consecutively under the keys of the mixes starting with that of the last mix. During the mixing operation, each mix "peels off" a layer of encryption by decrypting its input values with the last mix peeling off the last layer and outputting the original plaintexts. (...) In homomorphic mix-nets, each input ciphertext is a homomorphic encryption of the plaintext. During the mixing operation, each mix simply re-randomizes each encryption, resulting in a new ciphertext corresponding to the same plaintext. Eventually, the output tuple of the last mix need to be decrypted to retrieve the original plaintexts»*

Let us consider homomorphic mix-nets - so, mix-nets that work with reencrypting a message - as they can be accomplished with ElGamal(section 2.4 presents in detail how), and were used in previous versions of Helios, the protocol we want to analyze later.
We could idealize a voting system that would have voters produce their encrypted ballot ballot with the aid of a public-key (that was made available by an authority beforehand) and send it to the tallying authority. After making sure that voter was enlisted, the ciphertext could be published on a board alongside a voter-ID to guarantee the vote was **recorded-as-cast**. Authentication would not be needed if anyone could confirm that the said ID corresponded to a voter who was entitled to vote.
At the proper time all voter IDs can be separated from the encrypted votes, and the ciphertexts can be put through a mix-net that reencrypts the encrypted votes with the same public key producing new ciphertexts (unrelated to the original one) for each encrypted ballot, while mixing their order. As one authority would then still be able to relate the output of the mix to its input, various mixes could be devised, each reencrypting. If at least one authority works honestly in not divulging the mix, the system guarantees privacy.
These new ciphertexts could then be decrypted with the secret key that corresponds to the public election key - they would not be traceable to the original ciphertexts sent in by the voters - and the votes could be counted in a straightforward way.
But, one may ask, if trust cannot be guaranteed for all authorities and mix-nets eliminate the relationship between the ciphertext that is decrypted in the end and the one that was sent in by the voter, then how can one guarantee that an authority is not simply changing ciphertexts at will, to falsify the end result?
This and other problems we have so far overlooked will be addressed next.

2.3.6 Zero-Knowledge Proofs

There still are unsolved issues if we were to build an electoral system with the primitives we have considered so far. One was just put forward at the end of the previous sub-section. But is not the only one.
When using homomorphic ciphers to tally a vote one must keep in mind that a dishonest voter could encrypt a '2', instead of a '1', thus gaining two votes for the candidate of his choice. A dishonest or confused voter could also check multiple choices in an election where only one option is allowed. If ballots were to be individually decrypted, such misformatted ballots would be spoilt and not counted, but we are making a point of *not* decrypting individual ballots as that can be a security feature to guarantee privacy.
Also, when using threshold encryption a dishonest authority could publish a wrong public key share, or later when creating their decryption share a dishonest authority, interested in disrupting the result, could produce a fake share - it is a random-looking number, after all - and the final decryption of the homomorphic sum would not be possible, without giving away who is to blame. Any obligation that authorities make their secret keys known is violating the purpose of using threshold encryption in the first place.
What will help us here are yet another cryptographic "tools that allow you to prove that you have done a certain operation correctly, without revealing more than that fact"[5].
In an election setting, zero-knowledge proofs could be used for the following reasons[9]:

1. Trustees[12] are required to prove that they know the [secret] key that matches the public key they are publishing;
2. Trustees are required to prove that they honestly contribute to the tally of the elections;
3. Voters are required to prove the validity of the ballot they submit.

This significantly lowers the margin for errors or dishonesty from the authorities. If such proofs are provided, everyone can check that

1. Every authority holds the secret key that matches the secret key they published[13].
2. Every authority published a decryption share that is well computed from the homomorphic sum.
3. Every ballot is consistent with the rules of the election for a proper ballot.

We will sketch how these can work in an ElGamal environment in subsection 2.4.5.

[12] What we have been calling authorities

[13] This also is important because, in an ElGamal environment, a dishonest authority could, if it waits long enough, compute a public/secret key pair and publish his public key share as the division of this public key by the product of the shares of the other authorities. As Pereira[9] described, this would allow it to decrypt any vote individually. The authority would not know a secret key corresponding to the share he published, though.

2.4 ElGamal

ElGamal is a well-known public-key (or asymmetric) encryption system that offers homomorphic and threshold properties. We will show in this section how ElGamal works and demonstrate those properties. That will allow us to proceed to analyze Helios, a voting system built upon ElGamal.

2.4.1 The Basics

ElGamal was proposed by Taher Elgamal[14] in 1984[10]. As a public-key encryption system it uses a public key, *pk*, to encrypt a message and a secret key, *sk*, to decrypt. This means *pk* can be freely published by the holder of the corresponding *sk*. Anyone wishing to send a message can use *pk* to encrypt it and will not be able to decrypt it only by knowing the public key.
Rather than relying on permutation and substitution like systems before it, public-key cryptography works with mathematical functions to encrypt/decrypt messages[11]. One function, or algorithm, will take a key and encrypt the message and another function will be used with the corresponding secret key to decrypt. It is immediately obvious that knowledge of the public key cannot lead to knowledge of the secret key or of an encrypted message - even with knowledge of the algorithm used.
To keep notation consistent with Bernhard and Warinshi[5], we will follow their definition[15] of ElGamal:

> *Given* (p,q,g)*, pick sk as a random element of* $\mathbb{Z}_q$ *and compute* $pk = g^{sk}$*, mod p.*
>
> Encrypt(pk,m)*: Pick r at random from* $\mathbb{Z}_q$ *and set* $c = g * r (mod\ p)$*;* $d = m * pk^r (mod\ p)$*. Return* (c,d)*.*
>
> Decrypt$(sk,(c,d))$*: Compute* $m = d/c^{sk}\ (mod\ p)$

This works in a so called (p,q,g) group, the multiplicative cyclic subgroup of $\mathbb{Z}_p^*$ ($\mathbb{Z}_p$ without '0') with order q generated by g with p prime and $q = (p-1)/2$ also prime. In this group, multiplication modulo p is well defined and the cryptosystem is safe if discrete logarithms are difficult to find - it is believed they are.
Helios is an election scheme built on ElGamal using a ballot that, for a normal election[16], presents the voter with the options and so he can a box by the option (or options, in case multiple choice is allowed) he favours. The system then encrypts a 'one' for a ticked option and a 'zero' for an unticked one.

[14] The author's name has no capital 'G'.
[15] Their Definition 15
[16] Where the voter is presented with as many candidates as are running and picks one

2.4.2 Homomorphism

Remember how ElGamal encrypts/decrypts in a (p,q,g) environment:

$$\begin{aligned} \text{Encrypt}(pk,m) &= (g^r, m\cdot pk^r) = (c,d); \qquad [pk = g^{sk}] \\ \text{Decrypt}(sk,(c,d)) &= d/c^{sk} = m; \end{aligned}$$

Now let us see what happens if we have to different ciphertexts (c,d) and (c',d') and multiply them:

$$(c,d)(c',d') = (g^r g^{r'}, m\cdot pk^r \cdot m'\cdot pk^{r'})$$

Now if we decrypt the product of the ciphertexts:

$$\begin{aligned} \text{Decrypt}(sk,(c,d)(c',d')) &= (m\cdot pk^r\cdot m'\cdot pk^{r'})/(g^{r\cdot sk}\cdot g^{r'\cdot sk}) \\ &= m\cdot m' \end{aligned}$$

What we see here, then, is that by multiplying two different ElGamal ciphertexts, the result will be the encryption of the product of the original messages. This is a homomorphic property, but not a very useful one to build a voting system. What we would want is the sum of the original messages, so that the product of the ciphertexts, constructed as '0' or '1' to correspond to a mark on a ballot paper, would yield the *sum* of the votes. What we may notice, though, is that if we encrypt g^m instead of m, we will obtain a way to extract the sum of different messages m, m', etc, rather than their products:

$$\text{Encrypt}(pk,g^m) = (c,d) = (g^r, g^m\cdot pk^r); \qquad [pk = g^{sk}]$$

Now we will multiply two different ciphertexts and decrypt the resulting product:

$$\begin{aligned} (c,d)(c',d') &= (g^r\cdot g^{r'}, g^m\cdot pk^r\cdot g^{m'}\cdot pk^{r'}) \\ \text{Decrypt}(sk,(c,d)(c',d')) &= (g^m\cdot pk^r\cdot g^{m'}\cdot pk^{r'})/(g^{r\cdot sk}\cdot g^{r'\cdot sk}) \\ &= g^m\cdot g^{m'} \\ &= g^{m+m'} \end{aligned}$$

To get $m+m'$ out of this result is of course the same as solving the Discrete Logarithm Problem, which means it is very hard. For exponents in the range of the number of voters we can expect in a typical election, though, is should not be a big problem to try all possible natural exponents of g, for instance, to find the corresponding result and thus to extract the number "hidden" in the exponent.

2.4.3 Threshold Encryption

ElGamal also allows for key distribution in such a way that if the authorities do not cooperate to decrypt (and we will only want to decrypt the homomorphic sum of the votes) a ciphertext, no decryption is possible.
N-out-of-N threshold is pretty straightforward: Let all authorities generate an ElGamal keypair (pk_i, sk_i) in an agreed upon (g, p, q) group and share their public key share, pk_i, so that the universal public key, pk, can be computed as the product of all pk_i.
Now, to encrypt anyone can now use pk:

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); \qquad pk = \prod_{i=1}^{N} pk_i$$

To decrypt, every authority i uses its personal secret key share sk_i which has been kept secret, obtaining a decryption share d_i.

$$\text{DecryptShare}(sk_i, (c, d)) = c^{sk_i} = d_i;$$

The original message can the be extracted by diving the d part of the ciphertext tuple by the product of the decryption shares.

$$\begin{aligned} m &= d / \prod_{i=1}^{N} d_i \\ &= d / \prod_{i=1}^{N} c^{sk_i} = d / c^{\sum_{i=1}^{N} sk_i} = d / c^{sk}, \end{aligned}$$

which is the definition of decryption in ElGamal. This holds because:

$$pk = \prod_{i=1}^{N} pk_i = \prod_{i=1}^{N} g^{sk_i} = g^{\sum_{i=1}^{N} sk_i} = g^{sk}$$

2.4.4 Mix-nets

ElGamal has a straight way to implement homomorphic mix-nets as described in section2.3.5, which are the protocol in which each mix reencrypts the ciphertexts.
Lets take a ciphertext composed by the tuple

$$(c, d)$$

Using the definition of ElGamal we know that

$$(c, d) = (g^r, m \cdot pk^r)$$

where m is the message that was encrypted. Now if we use the same public key, pk, to re-encrypt

that ciphertext using randomness r' we will obtain ciphertext (c', d') like so:

$$\text{Encrypt}(pk, (c,d)) = (c \cdot g^{r'}, d \cdot pk^{r'}) = (g^r \cdot g^{r'}, m \cdot pk^r \cdot pk^{r'}) = (c', d'); \qquad [pk = g^{sk}]$$

This corresponds to the original message m being encrypted with randomness $r + r'$:

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

and can thus be decrypted with the secret key sk that corresponds to the public key used to encrypt and re-encrypt, pk.
So to construct a mix-net based on ElGamal, whoever runs a mix has simply to re-encrypt the ciphertext given as an input, and give as an output the re-encrypted ciphertexts in a different order: this way no ciphertext from the output can be traced to a ciphertext of the input and even ciphertexts that were originally published on a board are now irrelatable to a specific origin. This may be done several times by different authorities so that no one has to fear that someone can associate an output to an input - whoever is in charge of the mix may well be able to do so. As long as the same public key is used, in the end the final ciphertexts will be decryptable with the secret key or by doing threshold decryption.
This leaves space for dishonest mixers to substitute ciphertexts, though. In fact as ciphertexts are indistinguishable from random strings, no one can tell if an output ciphertext is a re-encryption or a totally different encrypted message, or indeed simply a random string from the ciphertext space. In order to avoid the need for trust here, zero-knowledge proofs can be used.

2.4.5 Zero-Knowledge Proofs

We will present three Zero-Knowledge Proofs that can be used with ElGamal to prove certain statements.

2.4.5.1 Schnorr Protocol

Suppose, in an ElGamal environment with parameters (p, q, g), we want to prove to someone that we hold sk that corresponds to a public pk. One protocol to do so, between a prover and a verifier, works like this:

1. The prover picks another key pair (pk', sk') and shows pk'
2. The verifier chooses a random value $c \in (0, ..., n-1), n \leq q$ and computes $pk'' = pk' * pk^c$
3. The prover can now compute and present sk'' because $pk'' = pk' * pk^c \Leftrightarrow g^{sk''} = g^{sk'+sk*c}$ and only knowledge of sk (the original challenge) allows computation of sk''.

Definition 2.4: Schnorr Protocol

This kind of interactive proof holds the form of an interactive Σ -protocol and one can immediately see it is not practical (being interactive) in a real-life application where, furthermore, such a proof only convinces one verifier: If Alice and Bob run the protocol, Charlie cannot be sure they did not collude and had Bob pick a sk'', with corresponding $pk'' = g^{sk''}$, then computing $pk' = pk''/pk^c$ and shared these values beforehand with Alice. Alice and Bob could still run the protocol, and Alice could produce sk'' in the end without knowing the sk that corresponds to the pk she made public in the beginning.
Therefore a transformation of this scheme is made in order to make it both non-interactive and universal:

> The Fiat-Shamir transformation of a Σ-protocol is the protocol in which Bob's choice of a challenge c is replaced by Alice computing the challenge as $c := H(y,b)$ where y is the value of which she is proving a preimage [the original pk] and b is her "commitment", the message that she would send to Bob immediately before getting his challenge. H is a cryptographic hash[17] function with range $\{0,1,...,(n-1)\}$ [5]

So in ElGamal, to prove that someone in fact holds the secret key to match its advertised public key, a Schnorr proof of knowledge is produced - with the Fiat-Shamir modification to avoid the need for interactiveness. A proof that one holds sk to a corresponding sk, then, is called π and consists of (y,b,c,d). It can be checked by confirming that $H(y,b)$ returns c and that $pk = b + c * y$.

2.4.5.2 Chaum-Pedersen

Chaum-Pedersen proofs can be used to proof that a decryption share in Threshold ElGamal was correctly produced. This means that in presence of the value a from an ElGamal ciphertext (a,b), and a public key share pk_i the value d corresponds to a^{sk_i} with sk_i your secret key share.
The protocol can be described as follows:

1. Inputs: ciphertext (a,b), public key share pk_i, secret key share sk_i.
2. Pick a random r from $\mathbb{Z}_q$. Compute $(u,v) := (a^r (mod p), g^r \text{ (mod p)})$
3. Compute a challenge as $c := H(pk_i, a, b, u, v)$
4. Let $s := r + c * sk_i$ (mod q)
5. Compute the decryption factor $d := a * sk_i$ (mod p)
6. Reveal d and the proof $\pi := ((u,v),s)$.[5]

Definition 2.5: Chaum Pedersen protocol

To verify, recompute that the hash yield c, such as $c := H(pk_i, a, b, u, v)$ and check if $a^s = u * d^c (mod p)$ and $g^s = v * (pk_i)^c$ (mod p).

[17] We will elaborate on these in section 2.5.1

2.4.5.3 Disjunctive Chaum Pedersen

To prove that a ciphertext encrypts a value in a given range or one of two possible values (e.g. either '0' or '1'), disjunctive Chaum Pedersen proofs can be used. In Helios they are used to proof an encrypted ballot contains either a '0' or a '1' and if proof is needed than in a list of options only one (or another condition) 1 is present, then a so called overally proof is submitted too. We will not elaborate on how to produce theses proofs and refer to [5] for that effect.

2.5 Other Primitives

As we will see further on, the primitives we looked at so far, are not enough for the analysis in our work. We will thus take a look at other systems that are not necessarily found in the literature as specifically used in voting systems, but rather find a general usage in all kinds of cryptographic protocols. As we will see, Certvote makes use of them.

2.5.1 Cryptographic Hash functions

A hash function H takes an input m of arbitrary length, and produces a fixed-size output h.

$$h = H(m) \tag{2.5}$$

Fundamentally we want $H(m)$ to be "an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either (a) a data object that maps to a pre-specified hash result (the one-way property) or (b) two data objects that map to the same hash result (the collision-free property)."[11]
Hash functions are used, as we will see in 2.5.3, in the construction of signature schemes, and we will also use suggest hash functions to guarantee ballot data has not been tampered with, when describing changes to Certvote.

2.5.2 RSA

Like ElGamal, RSA can be used to build an asymmetric encryption scheme - which means a scheme that uses one key to encrypt messages and a different one to decrypt ciphertext. In the particular case of RSA based schemes, both keys can be used to encrypt a message, which the other will decrypt - this property enables RSA to be used for signing messages, as we will see in 2.5.3.
On itself, RSA is a trapdoor permutation believed to be one-way: the secret key enables one to invert said permutation, but knowledge of the public key does only allow the regular permutation. To turn this permutation into an encryption scheme one usually uses random padding that will add randomness to plaintext, after which the permutation is used on the padded plaintext with

the public key, to encrypt. Decryption is achieved by using the secret key with the permutation, followed by unpadding to obtain the plaintext.
RSA was first publicly proposed by Ron Rivest, Adi Shamir, and Len Adleman in 1978 and is "the most widely accepted and implemented general-purpose approach to public-key encryption"[11]. Following [5], for a padded message block M and a ciphertext block C, we have:

$$C = M^e mod\ n$$
$$M = C^d mod\ n = (M^e)^d mod\ n = M^{ed} mod\ n$$

where $n = p * q$ is the size of the block[18], p and q are prime, $pk = \{e,n\}$ and $sk = \{d,n\}$. It is out of the scope of this work to show how d and e must be related for these expressions to hold but the essential requirements are the following[11]:

1. *It is possible to find values of e, d, n such that M^{ed} mod n = M for all M < n.*
2. *It is relatively easy to calculate M^e mod n and C^d mod n for all values of M < n.*
3. *It is infeasible to determine d given e and n.*

2.5.2.1 OAEP

One padding scheme used with RSA to build an encryption scheme is OAEP (Optimal Asymmetric Encryption Padding), which was defined by the RSA Laboratories in 2002 [12]. It makes use, internally, of a hash function and a mask generation function that parameterize OAEP. With proper choosing of these two functions RSA-OAEP is safe against chosen-ciphertext attacks[12].

2.5.3 RSA signatures

The idea behind digital signatures in general is to enable the author of a message to prove his authorship. By signing a message using his secret key, he creates a signature - usually of a fixed length - that can be verified with anyone with access to the signer's public key. Given a public key, pk and a secret key, sk, we can define, for a message, m, two functions as follows:

$$sign(sk,m) = \sigma \tag{2.6}$$
$$verify(\sigma,m,pk) = b, \qquad b \in 0,1 \tag{2.7}$$

If a given signature, σ, is valid for a message m and public key pk, then $b = 1$, else $verify$ will return $b = 0$.
To simplify we will write σ_{sk}^m to signify a signature σ for the message m calculated with sk.

[18] Which means binary blocks will be of i bits with $2^i < n < 2^{i+1}$

We will define, for the scope of this work, a signature function that returns the original message and the appended signature, like this:

$$sig(sk, m) = (m, \sigma_{sk}^{m}) \tag{2.8}$$

Definition 2.6: Signature

Following[11] we can describe a signature using RSA: for this one needs to use a hash function to create a fixed-size output for a message of any size. The hash is then encrypted with RSA using the signer's *sk*. Message and signature are then sent to the destination. The recipient - or anyone, for that matter - can then generate a hash for the message himself and decrypt the RSA encryption of the signature using the sender's *pk* - comparing the hash he computed and the one he decrypted, he can verify the signature if they are equal. As the sender's *sk* is only known to the sender, no one else could have produced that signature.

2.5.4 AES

Contrary to ElGamal and to RSA, AES (Advanced Encryption Standard) allows the construction of a symmetric cipher scheme, with only one key to encrypt and decrypt. The only key is to be kept secret, hence called secret key (*sk*), and has somehow to be distributed before the exchange of secret messages is to take place.
AES on itself is a pseudorandom permutation (or a symmetric key block cipher algorithm) over blocks of 128-bits. It was chosen (when used with modes of operation as we will see right away) as encryption standard by the National Institute of Standards and Technology (NIST) in 2001[13] after being introduced as 'Rijndael' by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, in the AES selection procedure. As "the structure of AES and most symmetric ciphers is quite complex and cannot be explained as easily as many other cryptographic algorithms"[11], we will refer the reader to the definition in [13] and to the analysis in [11] if a complete description and study is needed. Let it here be said that AES works over blocks of 128 bits, with a key size of 128, 192 or 256 bits. One will speak of AES-128, AES-192 or AES-256, depending on the key size. The function operates several rounds of "byte substitution, permutation, arithmetic operations over a finite field, and XOR with a key"[11] on the message.

2.5.4.1 Cipher Block Chaining mode

To turn block ciphers like AES into encryption scheme, modes of operation are necessary. They allow the encryption of messages larger than the block size, while ideally randomizing the output. NIST recommends "five confidentiality modes of operation for use with an underlying symmetric key block cipher algorithm: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR). Used with an underlying block cipher algorithm that is approved in a Federal Information Processing Standard (FIPS), these modes can provide cryptographic protection for sensitive, but unclassified, computer data"[1].

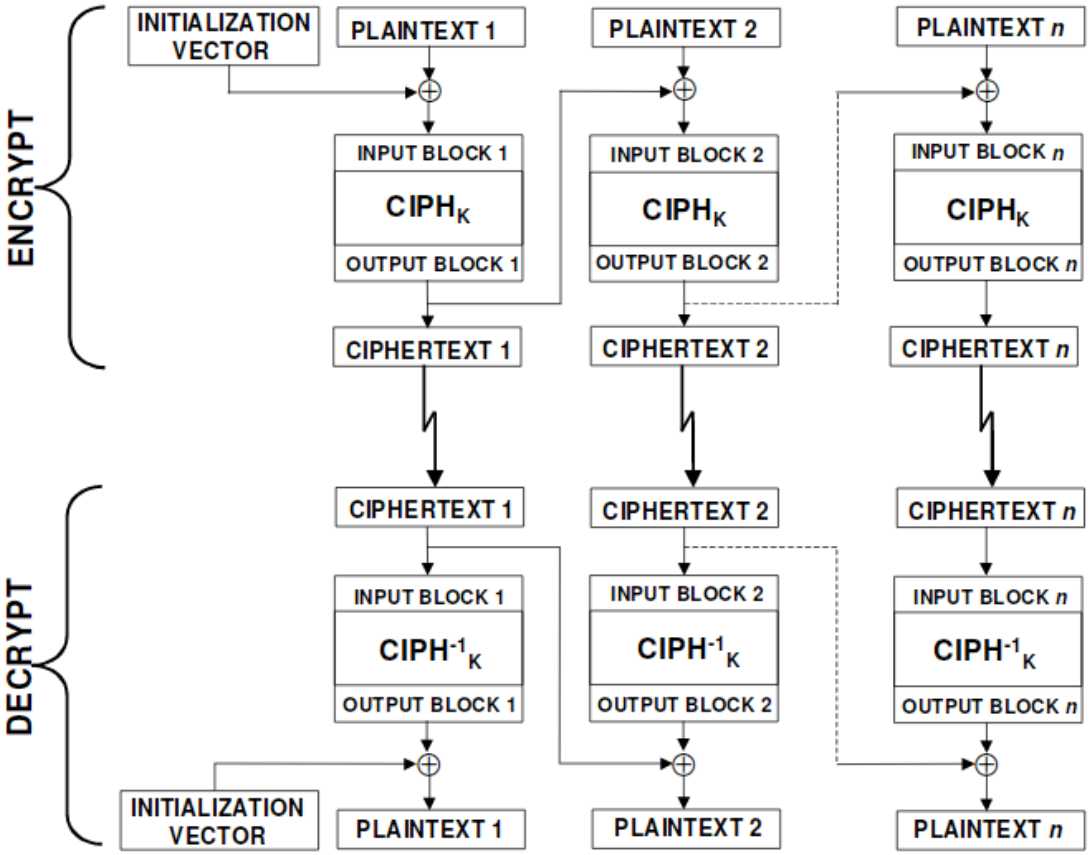


Figure 2.1: CBC-mode (from [1], p.10).

CBC mode, as shown in figure 2.1 works by "combining ("chaining") (...) the plaintext blocks with the previous ciphertext blocks. The CBC mode requires an IV [Initialization Vector] to combine with the first plaintext block. The IV need not be secret, but it must be unpredictable"[1]. AES-CBC produces messages that are safe from chosen-ciphertext attacks.

2.5.5 Shamir Secret Sharing

Rather than using threshold encryption to force collective decryption, it sometimes can be useful to share a secret - like a password, for instance - in a way that it only can be recovered when a certain number of people join with their secret shares. Again we can speak of a k-out-of-n share if any k parts of a secret divided in n are enough to reconstruct it.
In 1979, Adi Shamir[14] proposed a system that works built on the notion that given k different points (x_i, y_i) on the (x, y) plane "there is one and only one polynomial $q(x)$ of degree $k-1$ such that $q(x_i) = y_i$". For a secret D, represented as a number, one can then generate a random polynomial of the type $q(x) = a_0 + a_1 x + ... a_{k-1} x^{k-1}$ where a_0 equals D, k is the number of shares we wish to distribute and the pairs $[D_i, q(i)]$ represent those shares. Thus, with access to $k-1$ shares no information on D is gained and any k shares (k points of $q(x)$) allow to reconstruct $q(x)$ and find $D = q(0)$ if modular arithmetic is used, modulo a prime number p.

2.5.6 Hybrid Encryption

Public key cryptosystems are usually much less efficient than asymmetric ones. For that reason one can devise an asymmetric scheme for arbitrary messages that couples a symmetric cipher for arbitrary sizes messages (e.g. AES-CBC) and an asymmetric cipher for small messages (e.g. RSA-OAEP).

2.6 HSM

HSM (Hardware Security Modules) are dedicated pieces of hardware that store cryptographic keys and process cryptographic operations. Certvote uses a HSM located in Casa da Moeda, the portuguese mint. As per Annex D, Casa da Moeda offers high a security environment for sensible dat or operations. This includes separation from the Internet (LAN access only), sealed cableworks, restricted to data access on a logical level, exaustive loggin, contingency plans to ensure operation when natural, equipment, or human failure occurs, high-security instalations and traceability of every access and movement.
We will consider in this work that the HSM used in Certvote is tamper-proof and always works as configured.

2.7 Conclusion

In this chapter we have defined what essential properties an electronic voting system must present (Definition 2.1). We have taken a look on how paper-based elections work in Portugal, and how such elections fit into the properties we defined beforehand.
We then proceeded to describe the cryptographic characteristics that are referred in literature as useful when building electronic voting schemes, and we described ElGamal, a system that allows the access to said characteristics. We also introduced other schemes that will be important when analyzing Certvote, in the next section, or when introducing improvements in section 5.3.

Chapter 3

Certvote

Having defined the necessary tools, we will now take a look at the i-voting solution of Multicert, called Certvote.

3.1 Introduction

Certvote[1] is a voting solution produced by Multicert, a Portuguese company. Certvote exists for well over a decade now and took part in the Portuguese trials for electronic voting in 2004 and 2005. It has been analyzed by [15] but as we will see it has radically changed since then, being nowadays an internet voting system rather than one based on voting machines.
As an i-voting solution it has been served to private customers with success for the time of its existing.

3.2 Security Requirements

On their website, Multicert advertises the following properties[2] of the product.

> CertVote Warranties
>
> CONFIDENTIALITY OF THE VOTE CertVote ensures that the vote is secret and only the elements of the election commission can access the results after authentication.
>
> VOTING INTEGRITY There can be no change in the content of the vote without this being detected.
>
> ANONYMITY GUARANTEED Using cryptographic methods and hardware, your voters anonymity is guaranteed - it is not possible to trace the vote back to the voter.

These are essentially our properties of Privacy and Correctness from our Definition 2.1.

[1] https://certvote.com/
[2] http://certvote.com/#warranties

3.3 Description[3]

Certvote works with a voting commission (**VC**) that has n elements VC_i. The system can be configured so that k out of n elements have to work together to decrypt the final votes and thus tally the result. Key elements of the system are:

- the n VC_i members of the commission,
- the **mClient** web app that said members work with,
- the **Promoter** server that validates voters,
- the **Hardware Security Module (HSM)** that stores the election's secret key and decrypts the symmetric keys used to encrypt individual votes,
- the **'Sistema de Mediação de Voto'**[4] **(SMV)**module that stores encrypted votes, and
- the **mBallot** web app that voters log into to vote
- a **VOTE** which consists of an encrypted ballot (which signifies the voter's choices) and an encrypted key, all signed with the voter's signing key (as formalized in Definition 3.3.4.3).

Voter listings, the dates and further parameters of the election as well as the credentials for voter and VC elements are expected to be setup previously. We will assume all communication between computers is secure but not anonymous.

3.3.1 Setup Phase

To setup an election, all n elements of the Voting Commission will join at a computer to use their previously obtained credentials to login to the mClient web app. mClient will check the credentials with the Promoter server.
After all elements have successfully logged on, mClient will communicate this to the Promoter and an election setup can start. The Promoter will ask the HSM server to create a user for a random password generated by the Promoter. HSM will return the User Certificate for that user to the Promoter - this doubles as username for that user. The Promoter then generates a random alias for that election and asks the HSM to create a RSA 2048 keypair for that election. HSM will return the public key, $pk_{HSM}^{encrypt}$. The password for the user on the HSM and the election alias are then broken up into n parts with a Shamir Secret-Sharing algorithm to allow reconstruction with any k parts. The n parts are relayed back to the mClient app. There every member, VC_i, can download its part of the secret to keep to himself until the end of the election - after testing everything is consistent so far, that is. For that purpose the system then undergoes an audit phase.
To audit the process so far, the parties should now upload their secret share back to the mClient. It will then be sent to the Promoter that will reconstruct the secret using all n parts, obtaining

[3] This section is built on interviews with members of the development team at Multicert
[4] Vote Mediation System

the HSM user password and the election alias - the Promoter kept the User Certificate in the first place. It will then ask the HSM to encrypt a random string for that user and election, which will prompt the HSM to return a ciphertext. The Promoter will send that ciphertext back to the HSM, asking it to decrypt. If the HSM returns the original string the Promoter will consider the RSA encryption/decryption on the HSM is working properly and that the secret shares can enable proper access to the HSM. The VC members will receive an OK via mClient that everything is working as expected.

3.3.2 Voting Phase

When election time arrives, another webapp, mBallot, will be made available through web browsers on the internet. Voters can here present themselves with credentials they have obtained beforehand. After clicking "login" a RSA512 keypair will be locally generated and used to sign the credentials sent to the Promoter: the Promoter will get the user/password pair and the public RSA512 key (pk^{sign}_{voter}) just generated, signed with the secret RSA512 key (sk^{sign}_{voter}) that the local computer of the voter will keep. If the signature checks out and the user/password pair too, and if there is a valid election for that user underway, the mBallot will receive an authentication TOKEN alongside an 'OK' message that the voting can proceed.
The TOKEN consists of: authentication date, voter category, voter RSA512 public signing key pk^{sign}_{voter}, voter pseudonym[5], voting ballots which that given voter can access and election ID, all signed with a secret signing key, $pk^{sign}_{promoter}$, of the Promoter[6]. The signature scheme used by the Promoter is RSA-PSS with RSA2048 and SHA-1.
After the OK from the Promoter, the voter can ask for his ballots, with mBallot confirming in the TOKEN which ones he has access to. mBallot then sends the ballots and the election's public key ($pk^{encrypt}_{HSM}$) to the voter's computer. The voter can now proceed to select his options on the ballot(s) and generate an AES-CBC 192 symmetrical key, $sk^{encrypt}_{voter}$ (key which he will use to encrypt his options), and further encrypt the same key with the $pk^{encrypt}_{HSM}$ of the election using a RSA-OAEP scheme. He will sign this VOTE with sk^{sign}_{voter} and send it to mBallot. mBallot sends this VOTE with the TOKEN to the SMV, and the SMV:

- validates the signature on the TOKEN
- validates the signature on the VOTE
- validates if all ballots have been submitted
- validates with the Promoter if the Token is still valid[7]
- saves the votes (stripped of their signature) in his database if all checks out

[5] The pseudonym was generated with the voter list and is only know to the Promoter
[6] One might think of this TOKEN as a certificate by the Promoter validating the signature of the voter like a Certificate Authority would
[7] A voter may have logged in and voted on another session

- discards the TOKEN.

An 'OK' message will be sent back to the mBallot and back from here to the voter.
At the defined time, voting closes and no more votes are accepted.

3.3.3 Tallying Phase

k VC members log in to the mClient app from one computer. mClient validates the logins with the Promoter and acknowledges success/unsuccess.
The members then upload their part of the secret, which the mClient app sends to the Promoter who starts by checking all voting options in the database have zero votes, and then reconstructs the secret. A line of threads is set up, each of which:

- asks the SMV for votes, that are sent back[8] by the SMV to the thread on the Promoter - this is done by the order the votes entered the SMV.
- The thread then proceeds to send the encrypted AES key to the HSM, receiving in turn a decrypted $sk_{voter}^{encrypt}$.
- The SMV then decrypts the ballot with the key he just got. This goes on for as long as there are encrypted votes on the SMV

Finally the Promoter can write the final tally and make it public. The Promoter keeps no logs as to what it decrypted.

3.3.4 Stripped Down Description

It may be useful to look at Certvote from a more abstract point of view. We will for this, define some new primitives.

3.3.4.1 Token Request

During the tallying phase, mBallot generates a pair of RSA512 keys to use to sign the VOTE it sends to the SMV, who works as an urn. But as SMV does not hold the login details of the voters, the voter has to ask the Promoter to certify his public key. We may see this in this way:
Token Request is a primitive that takes as input a public signature key, a voter username and the corresponding password. It is signed[9] by the voter with the secret key that corresponds to pk_{voter}^{sign}, called sk_{voter}^{sign} and sent from mBallot to the Promoter. We symbolize "Token Request" as:

$$TR = sign(sk_{voter}^{sign}, [(pk_{voter}^{sign}), username, password)]) \tag{3.1}$$

Definition 3.1: Token Request

[8] A VOTE consisting of a ballot encrypted with an AES-CBC key, and that key encrypted with the RSA 2048 public key of the election
[9] Recall Definition 2.6

3.3.4.2 TOKEN

On getting a token request as defined in Definition 3.1, the Promoter will check if the signature on the request corresponds to the enclosed pk^{sign}_{voter}, if username and password are correct as stored in a previously established database and if an election is running for that user. He will then answer with a TOKEN containing the public key of that voter, the pseudonym of that voter and metadata on the election and the voter. This TOKEN is signed with the Promoter's secret signing key, $sk^{sign}_{promoter}$. A TOKEN, then, is defined as:

$$TOKEN = sign(sk^{sign}_{promoter}, [pk^{sign}_{voter}, pseudonym, metadata]) \tag{3.2}$$

Definition 3.2: TOKEN

3.3.4.3 Vote Construction

Upon receiving the TOKEN from the Promoter, mBallot will proceed to construct a VOTE. A VOTE consists of an encrypted ballot, signed with the secret signing key of the voter. Remember Definition 2.3 for an encrypted ballot, eB:

$$\begin{aligned} eB(key, B) &= \text{Encrypt}(key, B) \\ &= [(id_1, \text{Encrypt}[key, option_1]), ..., \\ &\quad (id_i, \text{Encrypt}[key, option_i]), ..., (id_n, \text{Encrypt}[key, option_n])]; \qquad option_i \in 0, 1 \end{aligned}$$

As we saw, Certvote sends the key that encrypted the ballot alongside with the encrypted ballot. So let us define encrypted key, $eKey$, as:

$$eKey = \text{Encrypt}(pk^{encrypt}_{HSM}, sk^{encrypt}_{voter}) \tag{3.3}$$

Definition 3.3: Encrypted Key

Now we can define a vote as:

$$VOTE = sign(sk^{sign}_{voter}, [eB, eKey]) \tag{3.4}$$

Definition 3.4: VOTE

So a VOTE is a signed tuple, consisting of an encrypted ballot and the encrypted (with the election's public key) key to that ballot.

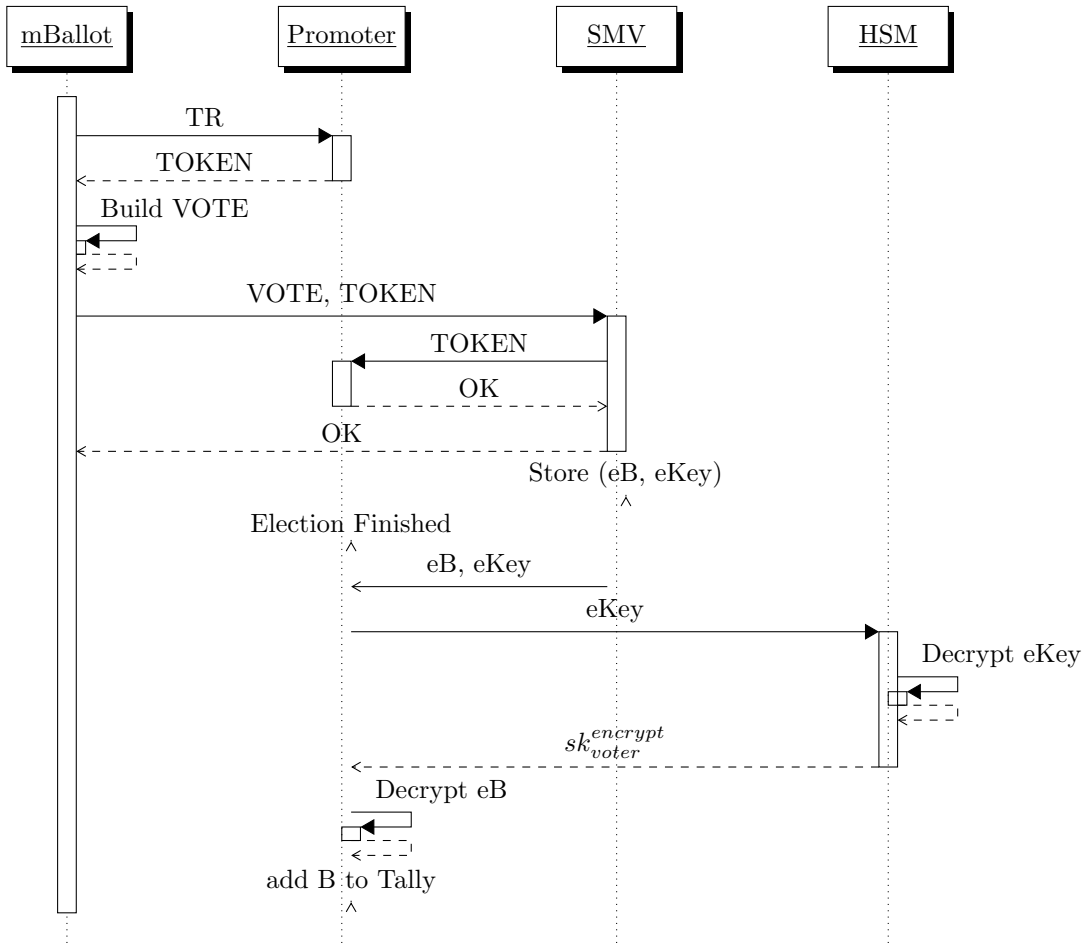


Figure 3.1: A stripped down view of Certvote.

3.3.4.4 Stripped Down Diagram

Figure 3.1 shows an abstract view of how Certvote handles a vote from the moment a user logs in until the vote is counted, showing how the different system components interact. It simplifies the workings of Certvote without changing the overall transit of sensible information the and encryption/decryption processes. We assume that all verifications (credentials, signatures) are ok, so a valid voter is interacting with a working system.

3.4 Analysis

We will now take a close look on how these workings place Certvote in respect to our findings.

3.4.1 Primitives

To guarantee **privacy**, Certvote has the votes of every user encrypted with a 192bit AES key ($sk_{voter}^{encrypt}$) in CBC mode. Said key is then encrypted with the election's RSA 2048 bit public key, $pk_{HSM}^{encrypt}$ with OAEP-SHA-1 and MFG1. $sk_{voter}^{encrypt}$ is only to be decrypted after the polls close,

and subsequently it can be used to decrypt the vote itself. It is essential then that AES-CBC and RSA-OAEP are well suited to guarantee privacy and we should notice that we need not only short-term protection during the time the election is running (although this is important and assures **fairness** as defined in Definition 2.1) but individual votes are expected to remain secret well after the election ends, ideally forever. In the worst case scenario in which the communication between voter and the system is opened to eavesdropping or the encrypted votes are somehow else accessed, the encryption of the votes should still hold.
We refer to [16] to help us with the characterization of the primitives and schemes used in Certvote in terms of security. Essentially this document characterizes a cryptographic system as either *legacy* or as suited for new (or *future*) use. Legacy systems may be in use at the moment, and are still suited in that case, but should be phased out in the near future in benefit of more long-term solutions. A system that is not even appropriate for legacy use in the terms defined by the paper, should be «replaced as a matter of urgency».
Following the reasoning used in [16], we will first look at the scheme used and then at the underlying primitive.

3.4.1.1 AES-CBC 128

CBC - Cipher Block Chaining is a well-known mode of operation for Block Ciphers. If the underlying block cipher is secure - AES in our case - then it is IND-CPA secure. This is of course enormously important in an environment where Plaintexts - Ballots - may be formatted in a publicly known way. The mode is not IND-CCA secure, though, but as noted in [5], "[f]or the purposes of building ballot private voting schemes, non-malleability is sufficient".
We note that AES-CBC is malleable.

IND-CPA security: Indistinguishability under Chosen Plaintext Attacks presumes an "attacker may chose any two messages, send them to the security game [10] and get an encryption of one of them back; a scheme is called IND-CPA secure if she cannot tell which message the security game chose to encrypt."[5]
IND-CCA security: Indistinguishability under Chosen Ciphertext Attacks makes the same assumptions as the IND-CPA game, while also allowing the attacker to submit ciphertexts to be decrypted - except for the ones he is trying to make a statement about.

Definition 3.5: IND-CPA and IND-CCA

Looking at AES as a primitive, [16] classifies 128 bits, and hence the 192 bits Certvote uses, as fit for future use, but recommends 256 bits for long term use.

3.4.1.2 RSA encryption

RSA does not rely on a mode of operation to form a scheme but does require on a padding mode. This is important to avoid that the deterministic property of the plain RSA algorithm allows chosen ciphertext attacks. Certvote uses OAEP with SHA-1 and MGF1.

[16] cites OAEP as safe for future use, SHA-1, however, should be updated. SHA-1 is only recommended for legacy use, but [16] stresses that it «(probably)» does not offer the 80 bit security they consider minimum for legacy applications. They explicitly «propose removing SHA-1 from applications as soon as possible».
As for RSA itself, [16] cites 1024 bits as the minimum for legacy only, but encourages 3072 for near term future and 15360 bits for long term future use.

3.4.1.3 RSA signature

The signature that the **Promoter** uses to sign the TOKEN, and thus is essential to authenticate a voter with the **SMV** is RSA-2048 with SHA-1.
RSA-2048, as seen just before in 3.4.1.2, is enough for legacy use but 3072 or even 15360 bits are recommended for future use, near term or long term, respectively. The update from SHA-1 we just noted when looking at OEAP also applies here.

3.4.2 Protocol

3.4.2.1 Trust Assumptions

Referring to Figure 3.1 we can sketch what assumptions we have to make to trust the system in terms of its security requirements (3.2) - which came down to what we defined as correctness and privacy.
To break Privacy, a vote will have to be decrypted by an entity that also holds the TOKEN, which allows to cross a vote with the identity of its voter. Privacy will be guaranteed:

- if mBallot[11] does not leak information on the vote OR
- if SMV and HSM do not collude to decrypt ballots, as the SMV has eB and TOKEN, and HSM holds $sk_{HSM}^{encrypt}$.

To break Correctness one could have the following scenarios:

- mBallot encrypts an invalid or different vote than the voter intended OR
- SMV registers a different ciphertext than he got from a voter
- Promoter counts 'wrong' in the tallying phase either sending wrong eKey to the HSM or altogether ignoring the inputs from HSM or simply altering the votes at the adding phase.

So to guarantee Privacy we would have to be sure that mBallot does not leak votes and that the HSM does not collude with the SMV do decrypt ballots and publish the associated TOKEN.
As for Correctness, if we could assume the HSM is tamper proof and if voters could be sure there vote is well built by mBallot and well registered on the SMV, we could guarantee Correctness only needing trust in the HSM.

[11] Publishing its code will make this less of an issue.

3.4.2.2 Verifiability

While Privacy and Correctness are advertised as "the" properties of Certvote, we must not forget the other properties of definition 2.1.
We can easily observe that the system does not offer any taste of verifiability, as defined in 2. Votes cannot be verified as cast-as-intended or recorded-as-cast, by individual voters, nor can they be verified as tallied-as-recorded by anyone. This adds up to a system that is neither individually nor universally verifiable.
While this does not mean the system is unsafe, *per se*, verifiability is an essential property in modern e-voting/i-voting systems: it helps to build trust and does a great deal to detect corruptions of the various system components - making them very hard if not impossible to go unnoticed.
So in the case of Certvote one cannot, in fact, rule out that a client-side attack is able to corrupt the encrypted vote in order to spoil it or even to cast a different vote than intended. Even if the voter could see the ciphertext after encrypting his vote, it is indistinguishable from a random string so a voter could never be sure what is encrypted. And allowing the voter to see ciphertext and the corresponding symmetric/private key is bad on principle: It corresponds to giving out receipts which paves the way to easy coercion.
But even if a voter could be sure his vote is correctly encrypted, he could not be sure it reached the 'urn' (on the SMV, in this system) uncorrupted, or that the 'urn' records it correctly without (intended or not) manipulation.
And last but not least, even a correctly recorded vote can be discarded, changed or otherwise tampered with in a malicious or corrupted tallying process.
With verifiability features, then, Certvote could offer guarantees to the integrity of the results. This would mean users or observers would **not need** to trust the workings of the different components - as well as the honesty of its operators and programmers - to have faith in the result.
We will show later that if Verifiability features can be implemented (and we will argue they can and show so), if the HSM can be trusted and if authorities do not collude to decrypt votes before an election closes, the system can fulfill the properties of definition 2.1 in a level comparable to the State of the Art in e-voting systems.

3.5 Conclusion

After describing and analyzing Certvote we can now summarize how it stands in terms of our definition 2.1:

- **Correctness**
  - **Completeness** - Trust in the workings of the system is needed.
  - **Soundness** - As votes are decrypted individually, bad voter behaviour does not affect the tallying - wrong votes are simply discarded.
- **Eligibility** - Implies trust in the system.

- **Fairness** - If the system works correctly, unless k authorities collude, there is no way to leak results.
- **Privacy** - Votes are protected by RSA 2048 while communicated and if the system works correctly, individual votes are not published. A corrupted computer the voter is on could leak votes, though. If Promoter and SMV collude, individual ciphertexts may be exposed.
- **Verifiability** - No kind verifiability is present.
- **Uniqueness** - Implies trust in the system.

Apart from these properties what we want to note now is that Certvote does not, while still being a well-behaving voting system, use any of the primitives we described in section 2.3 as being usually referred in scientific literature on electronic voting systems.
Therefore we will now take a look at a system that builds exactly on those primitives, to see how it works, what properties it offers, and what we can learn from it in order to potentially improve Certvote.

Chapter 4

Helios

Having described Certvote we will now take a look another real-life i-voting system called Helios, the self-entitled «first web-based, open-audit voting system»[3].

4.1 Introduction

Since first presented in 2008 Helios has been updated to version 3[1]. The main reasons to focus on this specific system in our work, are that Helios is well-documented and reviewed, it has been used in real-world elections and that it focuses on verifiability, which is a key property that can distinguish electronic voting systems from traditional paper voting.
The current, 3rd, version of Helios is well characterized in Bernhard and Warinshi[5], De Vries and Bokslag[6] and Pereira [9]. Surprisingly it is not very well characterized by the developers, apart from the verification specs on their website[2]. For the setup and workings we will thus keep to the description in the quoted papers, but not before we take a look on what Helios claims to offer.

4.2 The Claim

Helios wants to be «open-audit», by which Ben Adida means two properties: «*ballot casting assurance*, where each voter gains personal assurance that their vote was correctly captured, and *universal verifiability*, where any observer can verify that all captured votes were properly tallied»[emphasis in original][3]. These two properties are essentially *Individual* and *Universal Verifiability*, which we have compounded into the Verifiability property in Definition 2.1. Verifiability is indeed the central claim of the system:

> Helios is deliberately simpler than most complete cryptographic voting protocols in order to focus on the central property of public auditability: any group can outsource

[1] Helios official website, accessed November 2016
[2] See footnote 1

its election to Helios, yet, even if Helios is fully corrupt, the integrity of the election can be verified.

What Adida stresses very strongly is that online or postal voting is *inherently* open to coercion as attackers can be "looking over the voter's shoulder". He therefore concludes Helios, working over the web - allowing people to vote with their web browser from anywhere - should only be used for low-coercion elections «student government, local clubs, online groups such as open-source software communities, and others» - not because it is unsafe in itself, but because voting via the world wide web is inherently open to coercion.

4.3 Description

We present a top-level description of a Helios election.

4.3.1 Setup Phase

Authorities agree on the (g, p, q) parameters for the ElGamal environment they will use in an election. Each authority generates an ElGamal keypair, with a Schnorr proof. Someone, e.g., one of the authorities, combines the public key shares, pk_i to generate the global public key pk. The global key and the parameters are published on the website for the election.
Furthermore, decisions have to be made about the voters: if anyone can vote or if there are voting lists (that can be uploaded into Helios as csv lists) and how the voters' identities are handled.
After all the setup is done, an election fingerprint is generated. "This fingerprint is not simply served from the Helios server, but actually recomputed by the BPS as a function of all election parameters: election URL, ballot encryption keys, questions, answer rules, . . . This feature provides a safety measure for the voter and could also help detect a malicious Helios server or vote invitation when an independent BPS is used."[9] and the parameters are frozen.

4.3.2 Voting Phase

With a frozen election, voters can start to vote. (Let us consider a simple election with a list of candidates where every voter chooses one option only. But the election could be setup with multiple questions and allowing multiple answers per questions.) A voter will use the *Ballot Preparation System* (BPS) page to construct a ballot[3] with his vote.
When the voter has answered all questions the election allows, he can review his options while the system has already encrypted the vote and created and presented a ballot tracker number - a hash of the ciphertext[6]. If the voter wishes to proceed to submit the vote it can, but the system also allows one to audit the vote. This step helps to guarantee that the vote is *cast-as-intended*. What the BPS does in this case is present the voter with all the information used in the construction of the encrypted ballot and its tracker, including the randomness used in the encryption process. The

[3] But as implied just before «Helios has an open API, which means that anyone (voter, candidate, activist, . . .) could program a ballot preparation system (BPS) that can be used to submit ballots in any election»[9]

voter can now use Helios' ballot verifier or any other independent tool to verify that the information the BPS used in fact renders a hash as presented beforehand. As the request to audit is only made after the vote was encrypted and the hash was presented as a commitment to the encryption, an audit can help to weed out corrupted systems - it is never known what encryption will be audited. In internet voting this is highly critical as the voters are using their day-to-day computers to cast the votes, so integrity of these machines cannot be guaranteed.
Audited ballots are then spoilt by the BPS - to preserve **privacy**, as letting the voter know the randomness used in creating an encrypted ballot would be the same as giving-out receipts with the plaintext vote - by the system and the voter will be presented with a new commitment hash the voter can audit again or post.
Only after the voter decided to post does the BPS need again to connect to the internet. In fact, until this point, the BPS has no knowledge whose vote it is preparing, allowing also observers to test an audit ballots at any time from any working device and avoiding attacks on the BPS of specific voters or voting groups. The user could indeed disconnect his machine after his browser loaded the BPS with the election parameters and would only now need to reconnect to the internet. The BPS will then authenticate on the election server and the ciphertext will be sent-in - alongside with a cryptographic proof that the ballot is well-formatted, without giving away the contents of the vote. The ciphertext and proof are then published on the bulletin board[4] of the election, thus guaranteeing the vote was *recorded-as-cast*. For convenience the previously displayed ballot tracker will in fact be displayed, with the whole information accessible next to it.

4.3.3 Tallying Phase

When the election ends, after voting time finished, the election server can compute an encrypted sum of all the votes[5] thanks to the homomorphic property of the ElGamal cryptographic system used to encrypt single votes. Each authority now has to use its secret key share to produce a decryption share and a proof of correctness of said share. This can be done using Helios software or any other tool constructed for the purpose. Authorities can download the encrypted sum, compute the share offline without sharing their private key share in any way, and upload the decryption share. Once all shares are available, Helios - or, again: anyone else - can decrypt the sum and end results are public. All secret keys can, and probably should, at this point be destroyed as they have no further legitimate use.

4.4 Analysis

We have noted before, Helios is well-studied and analyzed in scientific literature, namely in [5], [9], [8] or [6] and building on these analyses we can straight away take a look at how Helios looks in terms of our Definition2.1.

[4] The board is public and requires no authentication so that it cannot be manipulated according to who is accessing it.

[5] And indeed so can any observer as all needed information is public.

Helios guarantees all votes are counted and even lets voters and observers verify it, as we will see in a moment, so **Completeness** seems guaranteed. Also, by using proofs for correct keys, correct decryptions and correct ballot construction, the system provides protection against bad behaviour, thus guarantees **Soundness**.
Eligibility will be verifiable if no voter aliases are used on the PBB, but public and well-known voting IDs: in this case, and as votes are made public on the board, anyone can verify only legitimate voters were allowed to vote. If anonymous IDs are used, these will need to be shared between the board and the authentication server: the board only accepts votes from IDs it knows are legitimate, so eligibility can only be broken if board and authenticator collude.
Counts of the vote cannot be leaked unless *all* authorities misbehave, so an election is expected to be **fair**. The same goes for decrypting individual votes: **privacy** is guaranteed unless all authorities collude but the server could leak identified ciphertexts. Depending on how voter IDs are handled on the board, votes could be linked to voter in the future, if decryption of individual votes becomes possible by breaking the encryption. In terms of coercion-resistance, the authors of Helios concede it is low in internet voting.
Verifiability is the key selling point of Helios. Indeed in guarantees votes are **cast-as-intended**, via individual audits, **recorded-as-cast**, via a public board, and **tallied-as-recorded**, via homomorphism. This means the system is **end-to-end verifiable**, **universally** and **individually** verifiable.
Uniqueness seems trivial to assure, as a given ID will only have one vote on the board.
This quick analysis is consistent with de Vries and Bokslag[6].

4.5 Attacks

Nevertheless, known attacks and exploits on Helios have been described.
We should refer at this point that attacks directed against corrupting or manipulating the ballot as well as producing a corrupt count should be detected by the properties that lead to end-to-end verifiability in this system, making it much harder to attack these systems. Systems without, for instance, cast-as-intended verifiability could be compromised so that the application that builds a ballot simply encrypts a totally different vote than the voter intended. This is not expected to happen in Helios, as we saw before.

4.5.1 Clash Attacks

An attack called *Clash Attack* has been described by Küsters et.al.[17] Their reasoning is that if one can compromise the randomization used in the creation of encrypted ballots, one can essentially give the same receipt to different voters if they vote equally.
Each voter will then see his vote on the PBB not knowing other voters will be looking at the same receipt taking it for *their* vote. New false votes can then be inserted for each repeated receipt, to the liking of the attacker.
If votes are published with the identity of the voter next to it, such attacks will be detected by finding the same receipt for different voters.

If pseudo-anonymous identifiers are used for voters, this attack only works if the creation of these aliases also can be compromised: if the same alias can be given to two different voters and the PBB is only made public after polls close, the same receipt can be given out twice and again votes can be substituted. This only works if the users with the same ID vote equally - which can be predictable in some cases like well-known party members in a national election, for example.

4.5.1.1 Clash Attacks in Helios

In Helios, the attacker would have to corrupt a PBS and control the generation of random numbers so that different voters would generate the same ciphertext when encrypting an identical ballot. This could be done by transforming the generation of random numbers for each ballot into a deterministic sequence. This way every ballot would get the same "random" number after the same number of audits. And some given vote with three audits, for example, would show the same ciphertext as the same vote with three audits from a different voter.
If Helios is used with voters' names or other public IDs, clash attacks are not possible: each voter will see his name on the PBB next to the receipt. Repeated receipts on the board are a sign of tampering, thus being detected. But if Helios is being operated with anonymous aliases then one can succeed if the creation of such aliases is also corrupted: different voters can be given the same alias when aliases are being distributed, and so different voters will appear with the same alias and receipt on the PBB - oblivious to each other if votes are published all together after polls closed.
If Helios is used with a PBB without any voter ID, or with voter's IDs detached on another not-relatable list, then the attack is straightforward: only corrupting the PBS is needed.
Küsters et.al.[17] offer solutions ("Countermeasures") that can be worked into to the protocol to avoid such attacks.

4.5.2 Kleptographic Attacks

In [8] we find the description of a type of attack that can target any encryption scheme which makes use of a random bits in addition to the actual input. Rather than aiming at changing results, as the just described Clash Attack, this so-called *Kleptographic Attack* targets privacy and tries to pass information through the ciphertext without being detected. Here the randomness would be manipulated to violate privacy. "For example, in one naive approach, the parity of [the ciphertext] could indicate a vote for republican or democrat."[18]. An attacker checking for the ciphertext in a public board could then gain information on the contents of the vote which could be even more telling if more complex approaches were used.

4.5.2.1 Kleptographic Attacks in Helios

Like in the Clash Attack, to target the randomness used in the encryption of the ballot, an attacker would have to compromise the BPS to influence the creation of ciphertexts. Rather than actually controlling the random data used, if one were only looking to control the parity of the ciphertext as described before the system would only need to generate successive genuine ciphertexts until

it found one that fits the parity the attacker wishes. [8] notes there is no complete solution to this problem.

4.5.3 Ballot Stuffing

If an election system can insert votes that were not cast by actual voters, one speaks of ballot stuffing[19].

4.5.3.1 Ballot Stuffing in Helios

As all information to create a Helios ballot is public, it would be trivial for a corrupted PBB to create votes that were not cast in the first place, thus impersonating absentee voters. If real names are used on the PBB a potential voter who never actually voted could detect his name on the board and complain. "Helios with Credentials" - Helios-C - has been proposed as a variant[19], in which a Registering Authority is introduced, that provides every voter with a key-pair for signing their votes. The public credentials are then published. If every voter has to use its private key to sign the vote, then a dishonest PBB cannot stuff ballots for lack of a private voter key - assuming the Registering Authority is not colluding - with which to sign.

4.5.4 Cryptographic problems in Helios

Bernhard et.al.[20] have shown that the way the Fiat-Shamir transformation is used in Helios does not offer enough guarantees when constructing proofs for valid ballots, allowing the encryption of an arbitrary number of votes.
They argue that two variants of the transformation exist:“Both variants start with the prover making a commitment. The strong variant then hashes both the commitment and the statement to be proved, whereas the weak variant hashes only the commitment. This minor change yields dramatically different security guarantees: in situations where malicious provers can select their statements adaptively, the weak Fiat-Shamir transformation yields unsound/unextractable proofs.” Helios should be used only with the strong variant, which is not the case as of version 3[5].

4.5.5 Other issues

Not really a proper attack but rather an inherent characteristic of using a PBB is that in the future votes could eventually be decrypted if the cryptographic tool used for encryption (in this case ElGamal) is broken in the future. We have noted this in section 4.4 and it means that measures to obfuscate the relation between the voters true identity and the one on the PBB should be taken. This will probably take a toll on universal verifiability.

4.5.5.1 Coercion-Resistance and Vote Selling

As noted before and by the developers of Helios themselves, i-voting systems cannot protect against coercion. Attackers can demand to "look over the shoulder" of voters to ensure a vote

is made in a given way. Helios does offer the possibility of a voter voting multiple times - with only the last vote counting, of course - so one can come back after being free from coercion. But as changes in the vote will show on the PBB this is only a small protection.
It should be noted, however, that this indeed is much less a difference to traditional paper voting in an urn than it used to be. Nowadays a coercer who has the same power we considered to coerce in an i-voting environment can, in a traditional election, easily demand that a coerced voter presents a picture of a filled ballot as proof he voted in the demanded way. This seemed less-likely a few years ago but, nowadays with cameras present all-but everywhere (like a cellphone) seems much more possible - and thus coercion-resistance in paper voting must also be considered to be lower that maybe traditionally expected.
What is made easier in i-voting systems compared to paper voting is the scalability of coercion attacks and even of vote selling. In traditional voting an attacker still needs a coerced voter to go to a polling station and to deliver personally a vote into an urn. Over the web this is much easier as the attacker only needs the credentials of the voter, being able to produce the vote by himself then. It seems to us this is easier to do on a larger scale than with paper vote. On the other hand this seems also to hold for postal vote that, as seen in section 2.2.1, is allowed in Portugal under certain circumstances and no issue of this kind is known. What would minimize this issue if voters would need not only a login/password pair (or some other digital credential) to authenticate, but also a physical token The portuguese ID-card comes into mind as it holds official digital certificates for the card holder. It is reasonable to assume that a coercer would find it more difficult to get the physical ID-card compared to just a login/password pair.

4.6 Conclusion

As we showed in this chapter, Helios presents itself as an end-to-end verifiable internet voting system, which lives up to its claims. Is relies on a lot of the cryptographic tools we presented in chapter 2 and allows voters to check if their vote is properly handled all along the way: when they cast it, when it is registered by the system and when it is counted. This will help make people trust the system while allowing for malfunctions or tampering to be detected easily. What at this point might be asked is if we could provide analogous sens of trust in a system like Certvote that does not make use of the cryptographic tools that make verifiability possible out of the box. We will try to do so in the next chapter.

Chapter 5

Suggestions on Certvote Development

With our analysis of Certvote in mind and taking into account what we have found in the literature we can now proceed to suggest changes to Certvote that will work in the existing protocol and will improve the characteristics in the terms defined in 2.1.

5.1 Second Analysis

When comparing Helios and Certvote, we can again see that Helios offers verifiability features that will allow to confirm an election result is honest even if Helios itself is not honest. In fact, Adida in [3] used the phrase "Trust no one for integrity, trust Helios for privacy": One has to trust Helios not to disclose the identity of voters, but even a dishonest Helios server cannot fake a result - at least if one is willing to ignore the problems presented in section 4.5. If the authorities behave honestly, then the contents of the vote is also never leaked.
Certvote must rely entirely on trust. If any of its components fails or is tampered with it cannot be immediately detected by a voter. One should point out that this is not different from what happens with traditional paper voting, which is also mainly built on trust. Voter identity might be leaked by a corrupted Promoter in collusion with a corrupted SMV but if the HSM is tamper-proof, only collusion of the authorities allows the leak of individual votes. In this respect Certvote is actually *stronger* than Helios. Helios has the votes registered by the same server that confirms the login credentials while Certvote separates these systems.

5.2 Changes on the Cryptographic Schemes in place

As for the strength of the encryption of the votes in Certvote's protocol, we have seen that the public-key scheme should be updated in the near future if it is to be considered safe for the future. This of course will take a toll on the speed of the decryption and consequently on the speed of the tallying, but contrary to **privacy**, speed is not a critical property - at least not in the context of this work.

So to reach what [16] calls security for "future use", one should update AES-192 to AES-256 and RSA-2048 to RSA-15360. This is impractical as key generation alone would take minutes on personal computers, making the election process cumbersome. A compromise must be found and as we will not suggest the publication of ciphertexts (as happens in Helios) it will probably be enough to use RSA-3072 to guarantee privacy even if encrypted votes are leaked.
The signature scheme in place, built on RSA-512 should not need as much protection - after all it does only need to hold for as long as the election is running - and so an update to RSA-1024 is recommended.
Also for both the signature scheme and OAEP, SHA-1 should be updated to SHA-256, which is considered safe for future use.

5.2.1 Malleability

As we noted in 3.4.1.1, ASE-CBC is malleable. This makes the system permeable to attacks where, if ciphertexts are somehow leaked, a third party could reuse the ciphertext, change it, and resubmit for instance to vote the same way or to vote the opposite way as the leaked voter.
In Helios, malleability of ElGamal (IND-CPA secure) is overcome by using Zero-Knowledge Proofs. Repeated proofs will be rejected by the system thus invalidating that ciphertexts were submitted as different, malleated, and already previously submitted ciphertexts. For use with AES, though, Zero-Knowledge Proofs can be quite complex.
As such, for Certvote we suggest turning the encryption of the ballots into an authenticated cipher. For this, a second 256bit key will be generated by the voter to apply an HMAC-SHA256 on eB. eB+HMAC would then be encrypted together with $pk_HSM^{encrypt}$. This would make any attack based on reproduction of ciphertexts impossible, in the event ciphertexts are leaked by guaranteeing, like in Helios, the non-malleability of the ciphers used.

5.3 Changes in the System

Before proceeding towards verifiability just a note on the way Certvote has the SMV send the $(eB, eKey)$ tuples to the Promoter during tallying phase 3.3.3. As indicated, this happens in the order the votes reached the SMV and it should not be. As the Promoter knows in which order it gave away TOKENs, a correlation of this with the order it receives the votes from the SMV could allow an attacker with control only over the Promoter to correlate vote with voter identity.

5.3.1 Implementing Verifiability

End-to-end verifiability is a noted advantage of electronic voting systems when compared to paper vote. Certvote would benefit from introducing verifiability features as they would obviate the need to trust local and remote computers for several aspects of the voting process, contrary to the actual system.

5.3.1.1 Cast-as-Intended

Cast-as-intended verifiability can be achieved in what looks an easily implemented way for the current system. Inspired by Helios, Certvote could implement a vote auditing feature for the voter to check independently if his vote is being properly encrypted. We sketch a suggestion:

- After receiving the ballots the voter chooses his options, the local machine then encrypts the vote with a locally generated AES-CBC 192 bit symmetric key, $sk_{voter}^{encrypt}$.
- The system next encrypts $sk_{voter}^{encrypt}$ with $pk_{HSM}^{encrypt}$, using RSA-3072 with OAEP and SHA-3 and MFG1. It presents the voter with a receipt as a commitment.
- At this point a voter can cast his vote. But he should also be able to ask his local application to reveal $sk_{voter}^{encrypt}$ as well as the randomness used in OAEP.
- With the knowledge of this information any voter could then proceed to recreate the commitment (the construction of which had been be made public) re-encrypt a ballot (ballot syntax should be public anyway) and to compare it with the information the application presented in the first place.
- A voter could follow this procedure for as long as he feels the need to using whatever tools possible. Every iteration would improve the trust in the ballot encryption scheme.
- The commitment of the ballot that is actually cast should be kept as a receipt, as we will see when discussing recorder-as-cast verifiability.

An implementation on these or similar terms severely reduces the need to trust local machines to produce proper votes. If malware targets a random one percent of voter's machines, then such an attack is probably detected if one hundred different voters audit their votes. In section 5.3.1.4 we will make a point that vote preparation and auditing should happen before authenticating.
As a proof of concept, we have programmed a little app in JAVA that produces an encrypted ballot and commits with a hash (defined in Definition 5.1), and then lets it be audited by publishing the relevant information (Figure 5.1). The corresponding audit app takes that information and the voting options, and reproduces the hash (Figure 5.2). A voter or an observer can compare both values and if they trust the audit app (which obviously should be located on another computer altogether), then they can trust their ballot construction system. Here $pk_{HSM}^{encrypt}$ is hardcoded, in a real-life environment it would either be pulled from an election server or the user could input it by hand.
If $\|$ represents concatenation, then we define receipt/commitment as:

$$receipt = H(\text{Encrypt}(pk_{HSM}^{decrypt}, sk_{voter}^{encrypt}) \| eB(key, B)) \tag{5.1}$$

Definition 5.1: Vote Receipt

Where H() is a safe hash function like SHA-3 (deemed secure for that purpose [16]).

Figure 5.1: Ballot construction

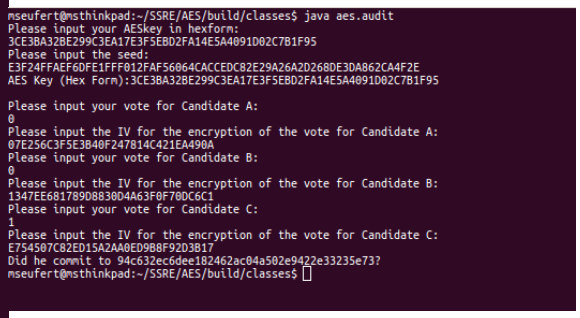


Figure 5.2: Auditing ballot

5.3.1.2 Recorded-as-Cast

After the ballot is cast and sent to the SMV, the voter needs an assurance it was also recorded in a proper way. To achieve this we again use Helios as an inspiration, to suggest that receipts should be published in a PBB of some sort[1].

To prevent clash attacks, described in 4.5.1, it is not enough to publish receipts without connection to user-IDs (either public IDs or pseudo-anonymous ones). It is reasonable to assume that some election rules will prohibit the publishing of public IDs (like name or Citizen ID-number) next to a vote receipt on a PBB, so we will suggest using pseudo-anonymous IDs. Certvote already uses an internal pseudonym when communicating (it is part of the TOKEN and not known to the SMV). If this could be made public to the voter when sending his vote, it could be the identifying part of the receipt.

So to avoid that a receipt gives away information of the vote, and to avoid clash attacks, the way to implement a PBB in Certvote would be to publish the receipt defined in 5.1 next to the pseudonym of the voter. Presenting a hash publicly does not leak information of the underlying hashed string - even if AES is broken or keys somehow. After polls are closed it will be impossible to reconstruct a ciphertext from its hash value.

A possible PBB, then, would work like this:

- The voter is presented with a commitment when building his ballot. After casting a ballot he should keep this for as a receipt verification purposes. Alternatively it could also be sent my e-mail or SMS, for instance, if the voter wishes. In any case, a digital signature should be provided, either on the website or via electronic message, so that a dishonest voter cannot later claim he was given a receipt that is not published. A digital signature gives the voter a guarantee that the receipt is genuine and avoids disputes afterwards.

- Certvote operates a publicly accessible website where it publishes a receipt for each vote it registered. Note that if this website is not public but requires authentication, verifiability is

[1] Let us remind here that these receipts can never give away the contents of the vote. This now becomes even more necessary if they are made public.

not strongly granted as the PBB could be generated differently to different users, defeating its purpose. A choice can be made as if it should work in real-time or only publish votes after polls have closed.

- Voters can then use their receipt to check whether the system has recorded their vote the same way it left the local computer. On the PBB the voter can search for the pseudonym only he will know, and then check if the hash corresponds to the one he was presented by mBallot.

- This opens the door to a type of coercion where a coercer demands a voter to go and vote. While no information on the contents of the vote can be gained, a coerced voter can be made to share a receipt with a third party who can then look for it on the PBB, particularly as the receipt is signed by the system. Such a trade-off is impossible to avoid.

5.3.1.3 Tallied-as-Recorded

To guarantee votes are tallied-as-recorded, Helios uses the homomorphic property to create a ciphertext that corresponds to the sum of all encrypted votes. This is not possible with AES encrypted votes.
We have to rely on the workings of the Promoter during the tallying phase. What could be offered as a verifiability feature, though, is the possibility to do recounts by a third party[2].
After results are computed and announced the state of the information is as follows:

- SMV holds the encrypted ballots and the encrypted keys.

- HSM holds the secret key that can decrypt the voters' keys.

At this stage it should not be hard to allow an auditor to present an application - either an open-code one, written by Multicert, or even one written by a third party and audited by Multicert - that proceeds to take the authorities' secret shares as inputs connecting subsequently to the HSM (in presence of the VC members and their secret shares) and the Promoter and redoes the tallying phase.

5.3.1.4 Final thoughts on verifiability

One further thing that would definitely increase trust in a system like Certvote would be the possibility to create encrypted ballots *before* authenticating as a valid voter. This would allow for external observers - who may or may not be voters - and any legitimate voter to audit the ballot preparation phase without the system knowing who it is dealing with. A party, an NGO or an

[2] In the Security Requirements for the Norwegian e-voting pilot process it is specifically noted that "the confirmation that the vote is correctly counted [may be] obtained in a controlled environment" and may "require a judicial process" https://www.regjeringen.no/globalassets/upload/krd/kampanjer/valgportal/e-valg/anskaffelse/ssa_u_appendix2b_requirements_table.xls, accessed January 2017

election observer could be creating and auditing ballots all the while the polls are open, thus ensuring the quality of mBallot and the code running on the voters machines. Also, a voter who does not feel comfortable with the system can ask for assistance and demonstration without sharing credentials or voting accidentally.
This would not need a great change in the current system, requiring only that a user could connect to mBallot and demand ballots for a given election and voter category. It would then, locally, generate the AES keys for encrypting ballots and construct the encrypted ballot with the voter's options. The system could then let a user proceed without authenticating to the audit phase.
Only when a voter felt confident to proceed to cast a ballot would the system demand authentication and then proceeding with the steps referred in 3.3.2 that had not been completed yet. This would also allow for all relevant cryptographic operations to happen offline: a voter should be able to download ballots and the elections public key from the system and unplug his computer from the Internet. Then he could create a ballot with his choices, create his AES key, encrypt the ballot, encrypt the symmetric key and only then reconnect to login and submit a the encrypted ballot and the encrypted key.
Attacks that target particular users or user groups are made impossible and external auditing is also made easier.
Finally we could devise a setup in which Certvote exists with an API that allows anyone to build a Ballot Preparation System and to use it to login and send ballots. This way political parties or other interest groups could provide people who trust them with an own interface to create encrypted ballots. Of course depending on how to implement this could again open the door for targeted attacks.

5.4 Improved Certvote

Now we can describe an updated version of Certvote's voting phase and tallying phase, with all changes in place (Figure 5.3[3]).

5.4.1 Voting Phase

As in the present version, mBallot is made available through web browsers, and any potential voter - but also non-voters such as observers - can access it and request to vote in an election. The election's public key $pk_{HSM}^{encrypt}$ is public, so is ballot syntax.
mBallot sends the requested ballots and the election's public key to the voter's computer. The voter can now proceed to select his options on the ballot(s) and

- generate an AES-CBC 192 symmetrical key, $sk_{voter}^{encrypt}$ (key which he will use to encrypt his options),
- generate a 256 key, sk_{voter}^{HMAC} and use it to generate an HMAC on the AES-CBC 192 ciphertext,

[3]Note the figure does not include vote auditing, as a vote that enters the system does not get to be audited.

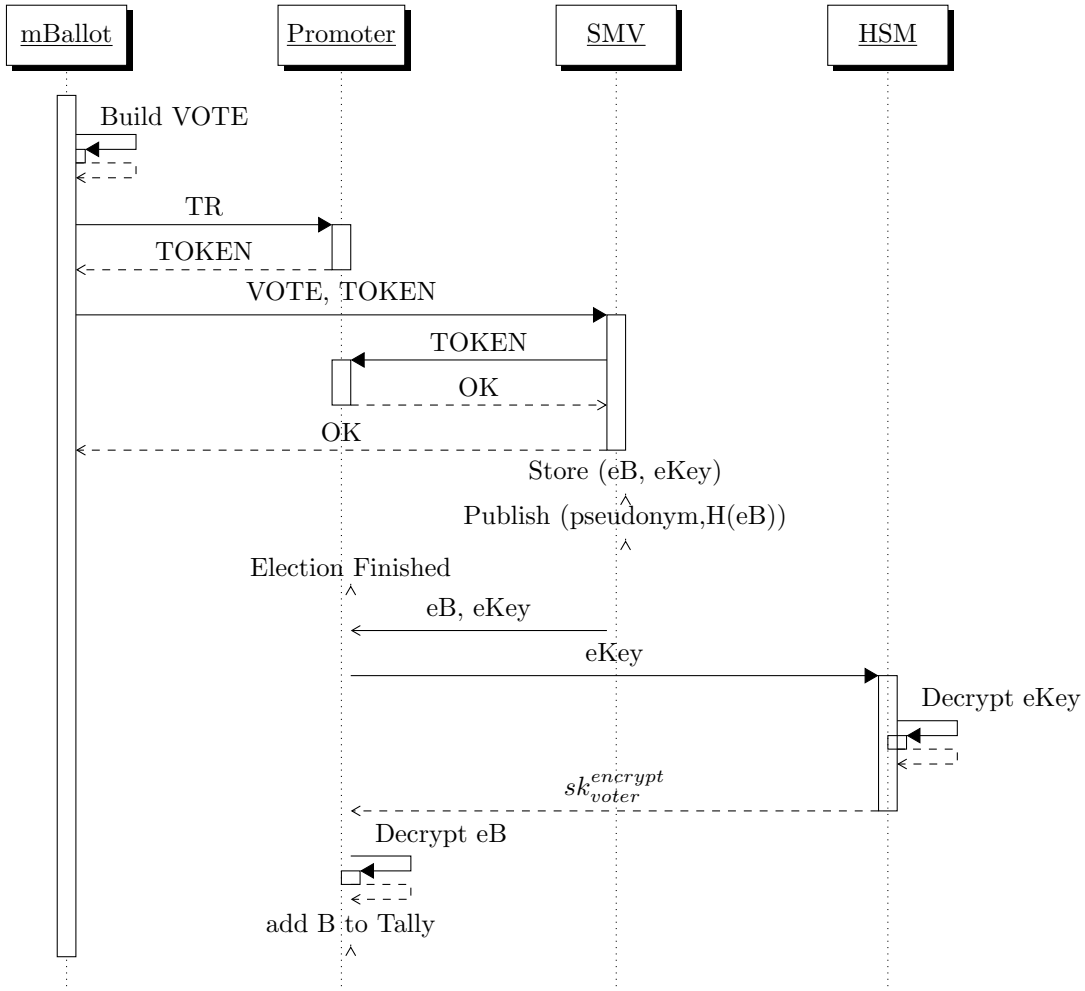


Figure 5.3: Improved Certvote.

- encrypt both secret keys with the $pk_{HSM}^{encrypt}$ of the election using the RSA-OAEP scheme.

A hash of the VOTE (as per Definition 3.4 updated with the HMAC) may serve as commitment and is presented to the voter.

5.4.1.1 Auditing Phase

At this point one may feel the wish to audit its vote, which the system allows. In this case the ballot plaintext is shown, so is the AES key used to encrypt it, the HMAC key and the relevant IVs (the RSA key is public already) and the randomness used in OAEP.
With this information, a voter can re-encrypt his ballot and reconstruct the hash to check whether the commitment was truthful. This renders the AES key useless and vote building must restart or else a coercer could demand this information a confirm a vote of the user on the public board.
The voter can repeat the auditing of his vote as often as he wishes.

5.4.1.2 Vote Casting

When ready to cast, he will have to login with his credentials, and should keep the commitment as a receipt as well as the Certvote's signature on that receipt. Before logging in, mBallot should destroy $sk_{voter}^{encrypt}$.
At this point mBallot will generate a pair of RSA keys for signing the Token Request and, as in the present version of Certvote, the Promoter will get the user/password pair of the voter and the public RSA1024 key (pk_{voter}^{sign}) just generated, signed with the secret RSA1024 key (sk_{voter}^{sign}) that the local computer of the voter will keep. If the signature checks out and the user/password pair too, and if there is a valid election for that user underway, mBallot will receive the authentication TOKEN alongside an 'OK' message that the voting can proceed. The TOKEN is the same as in the current system. The SMV signals mBallot that all is well and also shows the voter his pseudonym. After this, the SMV should publish the hash of the ciphertext, used as a commitment before and still a valid representation of the voter's intention, alongside the pseudonym of the voter.

5.4.2 Tallying Phase

After polls close, tallying would continue as it does in the present system. A hash of all the individual hashes on the PBB would serve as a commitment to the ciphertexts in the system.
A new feature would be the possibility to have an auditor recount the ballots.
Let us recall the relevant information for a recount:

- A list of ciphertexts stored on the SMV
- A secret Key on the HSM
- A public hash of all the hashes corresponding to stored votes, with a corresponding master hash of all hashes.

So an election independent election audit could work the following way:

- An auditor connects to the SMV in order to be able to ask for all encrypted ballots eB_i.
- Authorities introduce their secret shares to enable login on the HSM.
- Auditor connects and logs on the HSM.
- The auditor proceeds as Certvote did, for counting, but keeping a hash of every ciphertext.
- The auditor hashes all individual hashes.
- Compare tally and final hash to the original results. If they are the same, the election result was honest.

Note of course, that while recounting, the auditing algorithm has access to encrypted ballots, and proceeds to decrypt them. Also, when hashing the ciphertexts, the connection of that ciphertext to the information previously on the PBB becomes transparent. So if whoever is recounting cannot be trusted for privacy, then the auditing algorithm should be audited itself to make sure it does not leak information that can break privacy. The process can and should happen while disconnected from public networks and all information should be destructed after the tally is (re-)completed.

5.5 Helios and Improved Certvote

What we finally want to look at is how Improved Certvote parallels Helios not only in terms of the properties it offers, but also of the way it handles and protects sensible data, namely ballot contents.

5.5.1 Setup Phase

Let us assume both systems determine the authorities, and their credentials, the vote listings, and the voter's credentials, as well as the questions and further pre-voting information in the same way. Let us also assume the HSM on Improved Certvote is tamper-proof.
Both systems rely on a secret-share to divide the access to the final result and to the decryption of individual votes. Helios does so by having each authority create an ElGamal keypair and construct the public key from the individual public keys generated. In the case of Certvote we have the HSM that generates an RSA keypair, and never shares the secret key $sk_{HSM}^{encrypt}$. Instead, the access to the HSM, and thus to the possibility of decryption anything that the corresponding $sk_{HSM}^{encrypt}$, is shared between the authorities using a Shamir Secret Share.
At the end of the Setup Phase, thus, both systems present a public key for an asymmetric encryption scheme, with no corresponding secret key accessible by any individual authority.

5.5.2 Vote Phase

After voting is concluded, relevant information is stored as follows:

- In Helios: The PBB holds and publishes encrypted votes, alongside with voter alias, vote tracker and corresponding vote proofs.
- In Improved Certvote:The SMV stores encrypted ballots, with corresponding encrypted keys. Receipt and pseudonym for each encrypted ballot is made public.

The secret keys that, in both systems, enables decryption is not available unless authorities collude. The threshold property of ElGamal guarantees this in Helios, so does the HSM in Certvote. In both systems the votes that are stored were produced by verifiable methods. This means that if, in either system, the voting phase was well conducted then at its end, the information available has essentially the same degree of security.
In terms of privacy, if we consider the generation and distribution of alias in Helios and of pseudonym in Improved Certvote both were managed with the same degree of security, then privacy is equally safe in both systems. But even knowledge of the real ID behind alias/pseudonym does only inform a third party that a certain voter did cast a ballot (which amounts to the same information available from observing a polling station in paper voting) as neither ciphertext and proof (Helios) nor receipt (Improved Certvote) give out information about the contents of the vote.

5.5.3 Tallying Phase

When tallying is concluded, Helios offers the following information:

- The ciphertexts of all votes casted, and corresponding proofs of correctness.
- The homomorphic sum of said ciphertexts (although this can be reproduced at any time from the individual ciphertexts) present as a final ciphertext of the sum of the individual ballots.
- The decryption shares of every authority, with a corresponding proof of correct decryption.
- The final ciphertext decryption (although this can be reproduced at any time in presence of the final ciphertext and the decryption shares) that stands as the final result.

In Improved Certvote, the final information is not as public as this. We find:

- A public list of hashes with corresponding pseudonyms and a final hash of these hashes.
- A private list of encrypted ballots and corresponding encrypted keys on the SMV.
- $sk_{HSM}^{encrypt}$ secure on the HSM.
- The final result as an output of the Promoter.

It immediately becomes clear that the homomorphic and threshold properties of ElGamal in addition to the correct usage of ZKP allows for the result to be publicly (or universally) verifiable and reproducible at any time after polls close, without revealing any information of individual votes - as long as the ElGamal cipher used to encrypt individual votes cannot be broken and authorities do not collude to compute a master secret key.
In Improved Certvote verifiability of the result cannot be done publicly as decryption of individual votes would break privacy. This does not mean though, that it is not possible to verify the result. We have described a recount using a third party that can be trusted for privacy or, if this trust cannot be guaranteed, that operates in an audited environment that does not store nor communicate individual decryptions.

5.5.4 Final thoughts on Improved Certvote

What this analysis shows is that under certain assumptions, Improved Certvote behaves a lot like Helios, even though still working in the main constraints of regular Certvote, namely using its Hybrid Encryption scheme. But if the HSM can be trusted - and indeed HSMs are trusted for a great lot of security systems all around us - then we can say that Certvote mimics Helios quite successfully. So if Helios satisfies security criteria of a given user, then Improved Certvote probably does too. A full equivalency is only hindered because Improved Certvote does not allow tallied-as-cast verifiability without relying on a third party for recount, so individual voters of observers cannot perform this step individually.
One change to Certvote we have not suggested but which should be discussed for high-stakes election would be the introduction of a two-factor authentication when requesting a TOKEN. One could think, for instance, that apart from login/pasword, a user would need to use the security certificate from his ID-card to authenticate himself in the Promoter. This woul difficult vote selling, as voter would be less-inclined to share their card compared to only sharing their login details, and with proper logging on the Promoter side, would also make ballot stuffing impossible. The added complexity of using the card and not having access to a card reader at home could disencourage people from voting, though.

5.6 Conclusion

With these changes the system could improve from our analysis in Chapter 3 and offer the following analysis for Improved Certvote:

- **Correctness**
  - **Completeness** - The PBB together with a trusted recount guarantees all votes are counted.
  - **Soundness** - As votes are decrypted individually, bad voter behaviour does not affect the tallying - wrong votes are simply discarded.

- **Eligibility** - As in Helios, eligibility is guaranteed if registration of voters is done properly (so it is in paper voting): in this case the PBB guarantees only legitimate voters are considered.
- **Fairness** - If the system works correctly, unless k authorities collude, there is no way to leak results. If the HSM is inaccessible during voting (e.g. being disconnected), fairness is trivial to guarantee as no access to the secret election key is possible - even with collusion.
- **Privacy** - Votes are protected by RSA 3072, individual votes are not published. A corrupted computer on the voter side could leak votes, though. Building an encrypted ballot while offline, before even authenticating, lowers that risk severely, if $sk_{voter}^{encrypt}$ is destroyed before connecting. If Promoter and SMV collude, individual ciphertexts may be exposed in connection to the ID of the voter. In Helios this last leak is possible under the same fault assumptions.
- **Verifiability** - Individual Verifiability is present. Tallied-as-recorded verifiability implies trust in the external auditor.
- **Uniqueness** - PBB guarantees uniqueness.

We tried to show that Improved Certvote successfully mimics properties and trust assumptions of Helios and that consequently an analysis of Helios may be mapped on Improved Certvote. This is rather useful as Helios has been - and still is - analyzed by many parties that offered suggestions on how to improve its workings.

Chapter 6

Conclusion and Future Developments

6.1 Goals

The main goal of our work was to analyze Certvote, to characterize it in respect to the state-of-the-art solutions in cryptographic voting, and to suggest appropriate changes.
We have derived a set of properties from the literature and showed what cryptographic schemes are typically used to implement a system that tries to fulfill those properties. After describing and analyzing Certvote, we noted it takes a different cryptographic approach and was not designed to offer verifiability. Thus we tried to understand how a different system, built on the schemes we determined, works and implements the properties we obtained. For this purpose we chose Helios which is densely discussed in scientific literature while also used in real-world elections.
With the analysis on Helios as a starting point we proceeded to a second view at Certvote while indicating what changes could be implemented to achieve a better accomplishment of the defined set of properties. In particular we showed how verifiability could be implemented - a proof of concept for one particular solution was accomplished - and how the implementation of verifiability could bring Certvote up to par with Helios in terms of trust in a final result.
A point can be made that with the proper implementation, an electronic voting system can offer more assurances than paper voting, where verifiability is impossible.

6.2 Future Developments

Our suggestions in this work may be developed by Multicert and deployed into Certvote. In general the development of electronic voting solutions should be made *en par* with the scientific reasearch in the related fields. This has not only the effect of being on top of problems that might be identified but it also fosters a contact between scientific community and real-world system that seems crucial to the success of the latter.
Voting electronically can feel highly obscure to laymen (after all obfuscation is a key element of cryptographic voting) and researchers can play a crucial role in educating about the risks but also the guarantees an electronic system can provide. For instance the difference between a system as

we analyzed in this text and one based one electronic machines, to give an example, exists on so many levels it would take too long to elaborate. Yet this is probably not so obvious for the general public. Scientists can make this gap smaller by providing informed evidence.
If electronic voting is ever to play a part in Democracy, and not only in low-stakes elections, then electronic voting has not only to be safe in the terms we have defined, but it has to *feel* safe. It turned out that is also what this paper is about.

Appendix A

Presentation given at Multicert

Electronic Voting

Challenges, Building Blocks, and one example

Michael Seufert

MIEEC, FEUP

Multicert, Outubro 2016

Outline

- Electronic Voting Over IP
- As "Secure" as Paper Voting
- Cheaper, faster, more participation (?)

Paper Vote Properties

- Only registered voters get to vote
- Results are only available after closing
- Courts can order recount
- Nobody peaks at you in the cabin
- You don't get a proof of how you voted
- Nobody can force you to vote for a candidate

Challenges

Basic Challenges[1]

- Eligibility
- Fairness
- Privacy
- Correctness
  - Completeness
  - Soundness
- Unreusability

[1] (Fujioka et. al., 1992)

Verifiability[2]

Verifiability - End-to-End (E2E): voter and observer can verify all steps of the voting/counting process, thus precluding the need for trust

- Cast-as-Intended (voter)
- Recorded-as-Cast (voter)
- Tallied-as-Recorded (observer)

[2] (Ali and Murray, 2016)

Other Properties[3]

- Accountability/Auditability
- Usability
- Accessibility

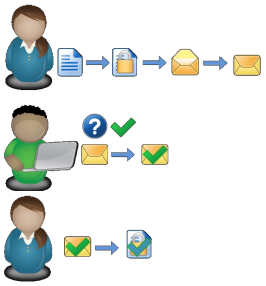
[3] (Ali and Murray, 2016)

Building Blocks

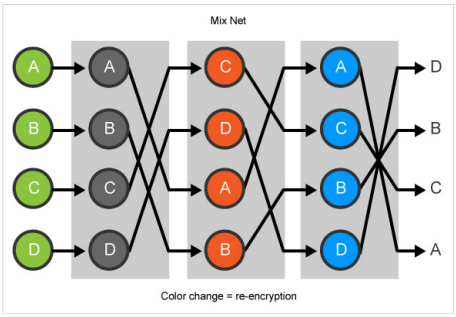
Building Blocks

- Blind Signatures
- Mixnets
- Homomorphic Ciphers
- Threshold Encryption
- Zero-Knowledge Proofs

Blind Signatures



Mixnets



Mixnets

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); \qquad [pk = g^{sk}]$$
$$\text{Decrypt}(sk, (c, d)) = d/c^{sk} = m \cdot g^{sk^r}/g^{r^{sk}} = m;$$

$$\text{Reencrypt}(pk, (c, d)) = (c \cdot g^{r'}, d \cdot pk^{r'}) = (g^{r+r'}, m \cdot pk^{r+r'})$$

Mixnets

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); \qquad [pk = g^{sk}]$$
$$\text{Decrypt}(sk, (c, d)) = d/c^{sk} = m \cdot g^{sk^r}/g^{r^{sk}} = m;$$

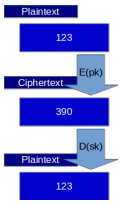
$$\text{Reencrypt}(pk, (c, d)) = (c \cdot g^{r'}, d \cdot pk^{r'}) = (g^{r+r'}, m \cdot pk^{r+r'})$$

Mixnets

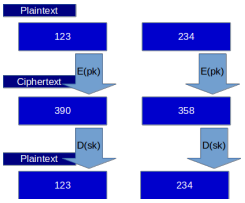
$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); \qquad [pk = g^{sk}]$$
$$\text{Decrypt}(sk, (c, d)) = d/c^{sk} = m \cdot g^{sk^r}/g^{r^{sk}} = m;$$

$$\text{Reencrypt}(pk, (c, d)) = (c \cdot g^{r'}, d \cdot pk^{r'}) = (g^{r+r'}, m \cdot pk^{r+r'})$$

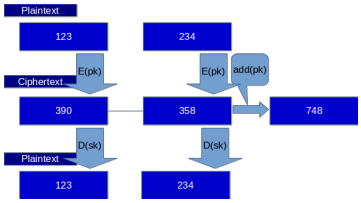
Homomorphic Ciphers



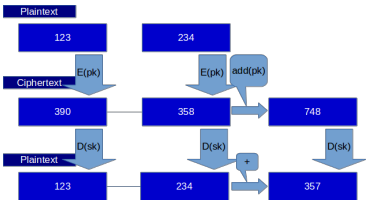
Homomorphic Ciphers



Homomorphic Ciphers



Homomorphic Ciphers



Homomorpic Cipers - for the geeks 1/2

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); \qquad [pk = g^{sk}]$$
$$\text{Decrypt}(sk, (c, d)) = d/c^{sk} = m;$$

$$(c, d)(c', d') = (g^r g^{r'}, m \cdot pk^r \cdot m' \cdot pk^{r'})$$
$$\text{Decrypt}(sk, (c, d)(c', d')) = (m \cdot pk^r \cdot m' \cdot pk^{r'})/(g^{r \cdot sk} \cdot g^{r' \cdot sk})$$
$$= m \cdot m'$$

Homomorpic Cipers - for the geeks 1/2

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); \qquad [pk = g^{sk}]$$
$$\text{Decrypt}(sk, (c, d)) = d/c^{sk} = m;$$

$$(c, d)(c', d') = (g^r g^{r'}, m \cdot pk^r \cdot m' \cdot pk^{r'})$$
$$\text{Decrypt}(sk, (c, d)(c', d')) = (m \cdot pk^r \cdot m' \cdot pk^{r'})/(g^{r \cdot sk} \cdot g^{r' \cdot sk})$$
$$= m \cdot m'$$

Homomorpic Cipers - for the geeks 1/2

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); \qquad [pk = g^{sk}]$$
$$\text{Decrypt}(sk, (c, d)) = d/c^{sk} = m;$$

$$(c, d)(c', d') = (g^r g^{r'}, m \cdot pk^r \cdot m' \cdot pk^{r'})$$
$$\text{Decrypt}(sk, (c, d)(c', d')) = (m \cdot pk^r \cdot m' \cdot pk^{r'})/(g^{r \cdot sk} \cdot g^{r' \cdot sk})$$
$$= m \cdot m'$$

Homomorpic Cipers - for the geeks 1/2

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); \qquad [pk = g^{sk}]$$
$$\text{Decrypt}(sk, (c, d)) = d/c^{sk} = m;$$

$$(c, d)(c', d') = (g^r g^{r'}, m \cdot pk^r \cdot m' \cdot pk^{r'})$$
$$\text{Decrypt}(sk, (c, d)(c', d')) = (m \cdot pk^r \cdot m' \cdot pk^{r'})/(g^{r \cdot sk} \cdot g^{r' \cdot sk})$$
$$= m \cdot m'$$

Homomorpic Cipers - for the geeks 2/2

$$\text{Encrypt}(pk, g^m) = (c, d) = (g^r, g^m \cdot pk^r); \qquad [pk = g^{sk}]$$

$$(c, d)(c', d') = (g^r \cdot g^{r'}, g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})$$
$$\text{Decrypt}(sk, (c, d)(c', d')) = (g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})/(g^{r \cdot sk} \cdot g^{r' \cdot sk})$$
$$= g^m \cdot g^{m'}$$
$$= g^{m+m'} \leftarrow \textit{Discrete Logarithm Problem}$$

Homomorpic Cipers - for the geeks 2/2

$$\mathsf{Encrypt}(pk, g^m) = (c, d) = (g^r, g^m \cdot pk^r); \qquad [pk = g^{sk}]$$

$$(c, d)(c', d') = (g^r \cdot g^{r'}, g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})$$
$$\mathsf{Decrypt}(sk, (c, d)(c', d')) = (g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})/(g^{r \cdot sk} \cdot g^{r' \cdot sk})$$
$$= g^m \cdot g^{m'}$$
$$= g^{m+m'} \leftarrow \textit{Discrete Logarithm Problem}$$

Homomorpic Cipers - for the geeks 2/2

$$\mathsf{Encrypt}(pk, g^m) = (c, d) = (g^r, g^m \cdot pk^r); \qquad [pk = g^{sk}]$$

$$(c, d)(c', d') = (g^r \cdot g^{r'}, g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})$$
$$\mathsf{Decrypt}(sk, (c, d)(c', d')) = (g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})/(g^{r \cdot sk} \cdot g^{r' \cdot sk})$$
$$= g^m \cdot g^{m'}$$
$$= g^{m+m'} \leftarrow \textit{Discrete Logarithm Problem}$$

Homomorpic Cipers - for the geeks 2/2

$$\mathsf{Encrypt}(pk, g^m) = (c, d) = (g^r, g^m \cdot pk^r); \qquad [pk = g^{sk}]$$

$$(c, d)(c', d') = (g^r \cdot g^{r'}, g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})$$
$$\mathsf{Decrypt}(sk, (c, d)(c', d')) = (g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})/(g^{r \cdot sk} \cdot g^{r' \cdot sk})$$
$$= g^m \cdot g^{m'}$$
$$= g^{m+m'} \leftarrow \textit{Discrete Logarithm Problem}$$

Homomorpic Cipers - for the geeks 2/2

$$\mathsf{Encrypt}(pk, g^m) = (c, d) = (g^r, g^m \cdot pk^r); \qquad [pk = g^{sk}]$$

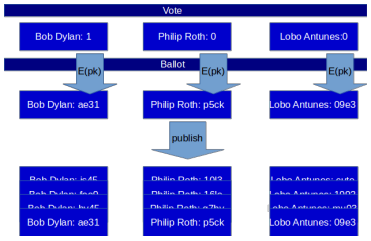
$$(c, d)(c', d') = (g^r \cdot g^{r'}, g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})$$
$$\mathsf{Decrypt}(sk, (c, d)(c', d')) = (g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})/(g^{r \cdot sk} \cdot g^{r' \cdot sk})$$
$$= g^m \cdot g^{m'}$$
$$= g^{m+m'} \leftarrow \textit{Discrete Logarithm Problem}$$

Homomorpic Cipers - for the geeks 2/2

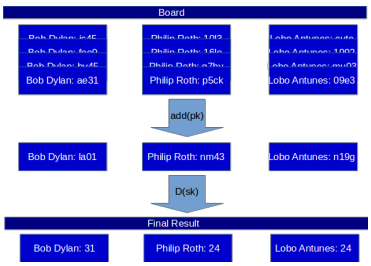
$$\text{Encrypt}(pk, g^m) = (c, d) = (g^r, g^m \cdot pk^r); \qquad [pk = g^{sk}]$$

$$\begin{aligned}(c, d)(c', d') &= (g^r \cdot g^{r'}, g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'}) \\ \text{Decrypt}(sk, (c, d)(c', d')) &= (g^m \cdot pk^r \cdot g^{m'} \cdot pk^{r'})/(g^{r \cdot sk} \cdot g^{r' \cdot sk}) \\ &= g^m \cdot g^{m'} \\ &= g^{m+m'} \leftarrow \textit{Discrete Logarithm Problem}\end{aligned}$$

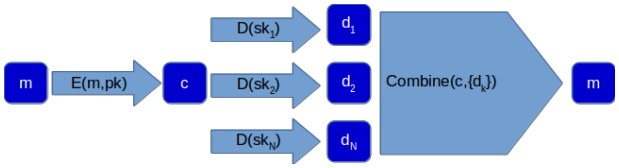
Homomorphic Vote



Homomorphic Vote



Threshold Cryptography



Thresholds for Geeks (N-out-of-N)

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); pk = \prod_{i=1}^{N} pk_i$$

$$\text{DecryptShare}(sk_i, (c, d)) = c^{sk_i} = d_i;$$

$$m = d / \prod_{i=1}^{N} d_i$$

$$= d / \prod_{i=1}^{N} c^{sk_i} = d / c^{\sum_{i=1}^{N} sk_i} = d / c^{sk}$$

$$pk = \prod_{i=1}^{N} g^{sk_i} = g^{\sum_{i=1}^{N} sk_i} = g^{sk}$$

Thresholds for Geeks (N-out-of-N)

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); pk = \prod_{i=1}^{N} pk_i$$

$$\text{DecryptShare}(sk_i, (c, d)) = c^{sk_i} = d_i;$$

$$m = d / \prod_{i=1}^{N} d_i$$

$$= d / \prod_{i=1}^{N} c^{sk_i} = d / c^{\sum_{i=1}^{N} sk_i} = d / c^{sk}$$

$$pk = \prod_{i=1}^{N} g^{sk_i} = g^{\sum_{i=1}^{N} sk_i} = g^{sk}$$

Thresholds for Geeks (N-out-of-N)

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); pk = \prod_{i=1}^{N} pk_i$$

$$\text{DecryptShare}(sk_i, (c, d)) = c^{sk_i} = d_i;$$

$$m = d / \prod_{i=1}^{N} d_i$$

$$= d / \prod_{i=1}^{N} c^{sk_i} = d / c^{\sum_{i=1}^{N} sk_i} = d / c^{sk}$$

$$pk = \prod_{i=1}^{N} g^{sk_i} = g^{\sum_{i=1}^{N} sk_i} = g^{sk}$$

Thresholds for Geeks (N-out-of-N)

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); pk = \prod_{i=1}^{N} pk_i$$

$$\text{DecryptShare}(sk_i, (c, d)) = c^{sk_i} = d_i;$$

$$m = d / \prod_{i=1}^{N} d_i$$

$$= d / \prod_{i=1}^{N} c^{sk_i} = d / c^{\sum_{i=1}^{N} sk_i} = d / c^{sk}$$

$$pk = \prod_{i=1}^{N} g^{sk_i} = g^{\sum_{i=1}^{N} sk_i} = g^{sk}$$

Thresholds for Geeks (N-out-of-N)

$$\text{Encrypt}(pk, m) = (g^r, m \cdot pk^r) = (c, d); pk = \prod_{i=1}^{N} pk_i$$

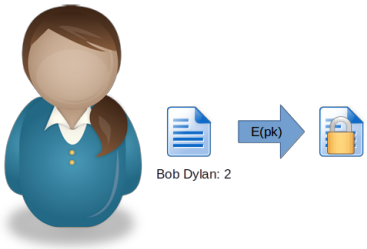
$$\text{DecryptShare}(sk_i, (c, d)) = c^{sk_i} = d_i;$$

$$m = d / \prod_{i=1}^{N} d_i$$

$$= d / \prod_{i=1}^{N} c^{sk_i} = d / c^{\sum_{i=1}^{N} sk_i} = d / c^{sk}$$

$$pk = \prod_{i=1}^{N} g^{sk_i} = g^{\sum_{i=1}^{N} sk_i} = g^{sk}$$

Dishonest Voter



Zero-Knowledge Proofs

- How to prove voter encrypts 0 or 1 and not, e.g., 2 to get an advantage for one candidate?
- How to be sure an authority holds her secret key?
- How to trust the announced result is correct?

Zero-Knowledge Proofs[4]

[4] (Bernard and Warinschi, 2014)

One Example

We present Helios v3 as described by (Bernard and Warinshi, 2014) and (de Vries and Bokslag, 2016) for a first-past-the-post election. Authorities use n-out-of-n threshold keys, .

Helios preparation

Each **authority** generates its ElGamal pk_i/sk_i pair, with Schnorr proof of knowledge π_i for the public key. One authority then combines the universal public key and publishes it along with the all pk_i and (p, q, g).

Ballot preparation

The **voter** can then get the public key and check the correctness of the Schnorr protocol as well as of the key combination. For each candidate the voter will be presented a box labelled correspondingly. In each the Ballot Preparation System will either encrypt g^0, for no, or g^1 for yes using the public-key and some randomness. Additionally a Fiat-Shamir proof proving it is either a 0 or a 1 is computed and added.

Ballot preparation

The **voter** can then get the public key and check the correctness of the Schnorr protocol as well as of the key combination. For each candidate the voter will be presented a box labelled correspondingly. In each the Ballot Preparation System will either encrypt g^0, for no, or g^1 for yes using the public-key and some randomness. Additionally a Fiat-Shamir proof proving it is either a 0 or a 1 is computed and added.

Ballot preparation

The **voter** can then get the public key and check the correctness of the Schnorr protocol as well as of the key combination. For each candidate the voter will be presented a box labelled correspondingly. In each the Ballot Preparation System will either encrypt g^0, for no, or g^1 for yes using the public-key and some randomness. Additionally a Fiat-Shamir proof proving it is either a 0 or a 1 is computed and added.

Pre-Cast Auditing

The **voter** gets a hash of the ciphertext and can, wishing so, proceed to audit the vote.

"Voting" and counting

When the **voter** feels confortable he can send a ballot to the counter (in Helios a webapp that tipically requires authentication) where it is published along with the voter-ID. All steps before this can be taken by anyone, namely non-voters. After closing the **authorities** check all proofs and reject invalid ballots. They should also reject ballots that are copies from earlier posted ballots.One authority can now sum the ciphertexts for each candidate and post that sum. Every authority produces a decryption share for each sum-ciphertext and posts it along with a DCP proof.Finally the tally can be completed by decrypting the sum and posting the result.

"Voting" and counting

When the **voter** feels confortable he can send a ballot to the counter (in Helios a webapp that tipically requires authentication) where it is published along with the voter-ID. All steps before this can be taken by anyone, namely non-voters. After closing the **authorities** check all proofs and reject invalid ballots. They should also reject ballots that are copies from earlier posted ballots.One authority can now sum the ciphertexts for each candidate and post that sum. Every authority produces a decryption share for each sum-ciphertext and posts it along with a DCP proof.Finally the tally can be completed by decrypting the sum and posting the result.

"Voting" and counting

When the **voter** feels confortable he can send a ballot to the counter (in Helios a webapp that tipically requires authentication) where it is published along with the voter-ID. All steps before this can be taken by anyone, namely non-voters. After closing the **authorities** check all proofs and reject invalid ballots. They should also reject ballots that are copies from earlier posted ballots.One authority can now sum the ciphertexts for each candidate and post that sum. Every authority produces a decryption share for each sum-ciphertext and posts it along with a DCP proof.Finally the tally can be completed by decrypting the sum and posting the result.

"Voting" and counting

When the **voter** feels confortable he can send a ballot to the counter (in Helios a webapp that tipically requires authentication) where it is published along with the voter-ID. All steps before this can be taken by anyone, namely non-voters. After closing the **authorities** check all proofs and reject invalid ballots. They should also reject ballots that are copies from earlier posted ballots.One authority can now sum the ciphertexts for each candidate and post that sum. Every authority produces a decryption share for each sum-ciphertext and posts it along with a DCP proof.Finally the tally can be completed by decrypting the sum and posting the result.

Helios' verifiability

- The voter can verify the ballot's correctness by auditing as much ballots as it takes to reach a certain probability. This assumes either trust in a third party verification application or a tech-savy user.
- The voter can verify his ballot is on the board and hence will be counted.
- Anyone can verify the added ballots render the anounced results.

Helios' verifiability

- The voter can verify the ballot's correctness by auditing as much ballots as it takes to reach a certain probability. This assumes either trust in a third party verification application or a tech-savy user.
- The voter can verify his ballot is on the board and hence will be counted.
- Anyone can verify the added ballots render the anounced results.

Helios' verifiability

- The voter can verify the ballot's correctness by auditing as much ballots as it takes to reach a certain probability. This assumes either trust in a third party verification application or a tech-savy user.
- The voter can verify his ballot is on the board and hence will be counted.
- Anyone can verify the added ballots render the anounced results.

Issues with Helios

- Coercion
- Public Votes
- Ballot Stuffing
- Clash Attack[5]

[5] Küsters, Truderung and Vogt, 2012

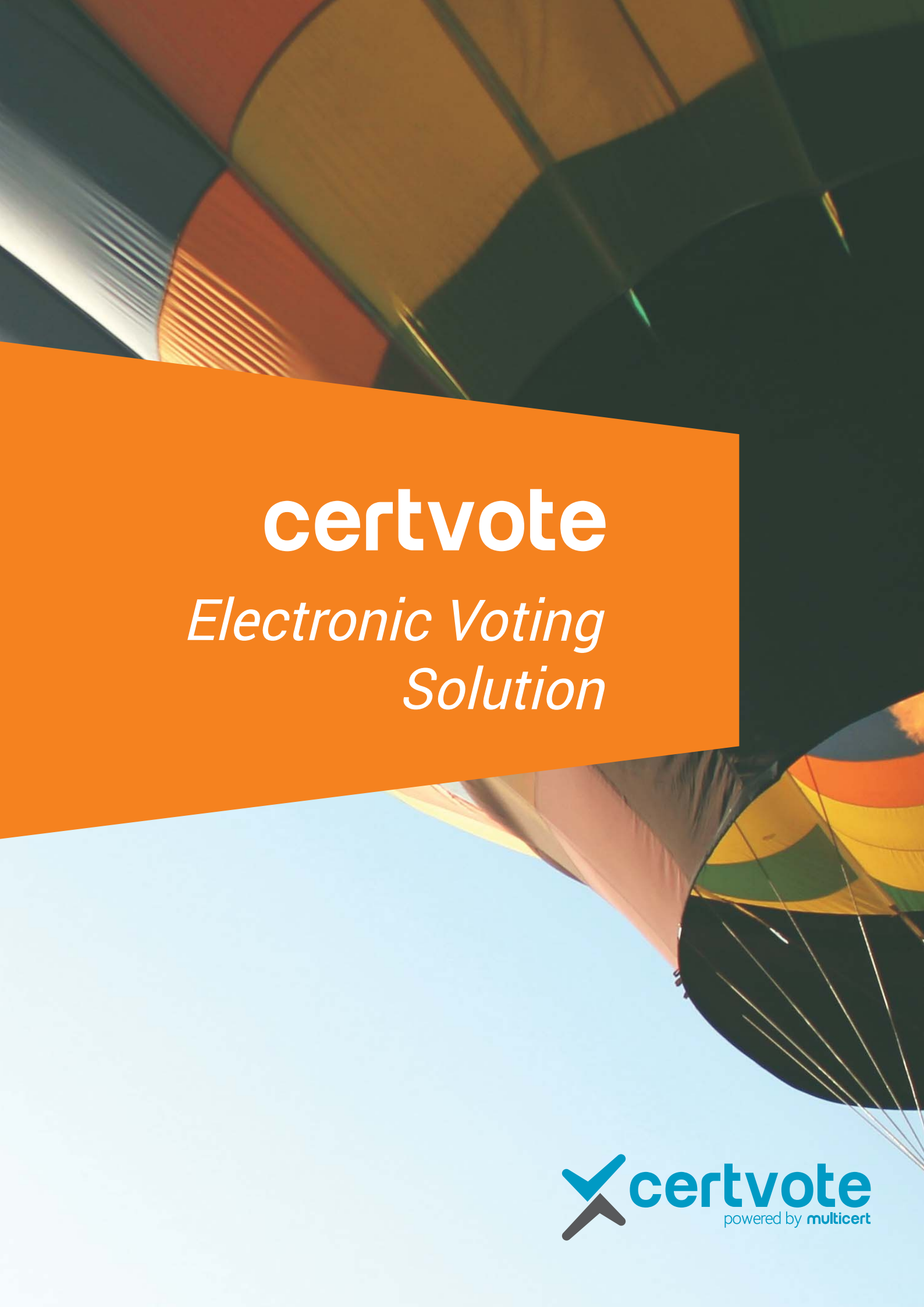
References

- Bernard and Warinschi, 2014: Cryptographic Voting—A Gentle Introduction
- Fujioka et. al., 1992: A practical secret voting scheme for large scale elections
- Ali and Murray, 2016: An Overview of End-to-End Verifiable Voting Systems
- de Vries and Bokslag, 2016: Evaluating e-voting: theory and practice
- Küsters, Truderung and Vogt, 2012: Clash Attacks on the Verifiability of E-Voting Systems

Appendix B

Certvote Product Brochure[1]

[1] Source: http://www.multicert.com/media/3785/multicert_prve_6.pdf, accessed November 2016



certvote

Electronic Voting Solution

certvote
powered by multicert

CertVote: Trusted Secure Platform

CertVote is the electronic voting solution from **Multicert**. It is a trusted secure platform, where a voting can take place exclusively electronic or in combination with the traditional paper ballots and postal voting. The platform guarantees the anonymity of the voter, since it is not possible to trace back who voted in what.

Two versions available: the **standard**, where you can configure and operate the voting autonomously through intuitive interfaces and easy to follow wizards, and the **premium**, which is customizable either by our team or by the partner reseller, adapting the **CertVote** to the needs of each request.

Electoral Roll Management

CertVote features a web-based back office for Electoral Roll management. In combined electronic and in-person voting acts, selected officers can search for voters, check and change the status of a voter, discard the electronic vote if the voter is overriding it with a paper ballot, resend access credentials through SMS and access selected voting statistics (e.g. voter turnout).

Several Voting Types

CertVote supports hybrid statutory election provisions, allowing an electronic voting to be conducted simultaneously with in-person voting and postal voting.

Easy and Intuitive to Use

CertVote allows elections' creation and management through a easy of use and vey intuitive backoofice. It also allows access to reports and statistics during the election.

Electronic Vote Casting

Electronic vote casting is very practical, simple and intuitive. It enables users to exercise their right to vote by using a computer interface, where the voting ballots are presented on screen. This component allows the voter to verify and confirm the chosen options before casting the vote by presenting a summary page. In addition to standard PCs, voters may also access this interface on mobile devices, such as smartphones and tablets.

To increase security to the best industry practices, we use technologies that cyphers the data and communications end-to–end. The voting process also has a time limit to prevent unauthorized access if the user becomes absent.

Voting Results

Voting Results component is where the initialization and closure of the voting process is performed by the voting committee members. After closure, the automatic counting process is initiated and the final results are presented and exported to several formats (CSV/Excel, PDF, DOC, XML). The results can be presented by groups/regions/sections, depending on the aggregation level needed.

Who can benefit from the CertVote solution?

» Listed Companies

» Unions

» Associations

» Professional Bodies

» Sport Clubs

» Political parties

» Governments

CertVote Features

» Allows the authentication by username/password and/or organizational credentials + PIN code

» Sensitive data stored in dedicated cryptographic hardware, non-exportable

» Multilingual

» Support and tutorials

» Web back-office for administration and operation

» Wizards for self-configuration

» m-of-n scheme for secure and reliable secret sharing between voting committee members

» Automated counting

» Customizable ballots – logo, options, text

» Supports several voting rules – blanks, null, multiple choice, voting shares

» Audit logs

Why Multicert

» Developing security turnkey solutions since 2000

» 1st private CA accredited by Portuguese National Security Office for issuing qualified certificates

» Deployment of large scale PKIs for eID and ePassport in Portugal and Cape Verde, issuing more than 4 million certificates/year

» Broad range of security solutions, developed by Multicert

» Highly skilled team

» Track record of international projects

» Quality and excellence: ISO 9001, ISO 27001, CMMI level 3

About Multicert

We are a supplier of end-to-end security and digital certification solutions for all types of electronic transactions that require security (e-commerce, e-banking, e-government, e-mail, amongst others). We operate across the value chain of the electronic certification business, and have skills in development, production and marketing of solutions that guarantee the security of electronic exchanges.

Multicert range of products and services covers not only the issuance of digital certificates (web servers, individual, codesigning, application, etc.), but also in PKI (Public Key Infrastructure) systems to businesses and Government entities. In addition to both Portuguese and Cape Verde Citizen Card and Electronic Passport, we also work in solutions for Electronic Invoice, Electronic Vote, Electronic Postmark and EMV-CAP compliant 2-Factor Authentication.

Specializing in new information and communication technologies, we have started our activity in 2002 with a shareholder structure composed by SIBS, CTT, INCM and PT and we have offices in Lisbon and Porto, Portugal.

info@multicert.com

www.multicert.com

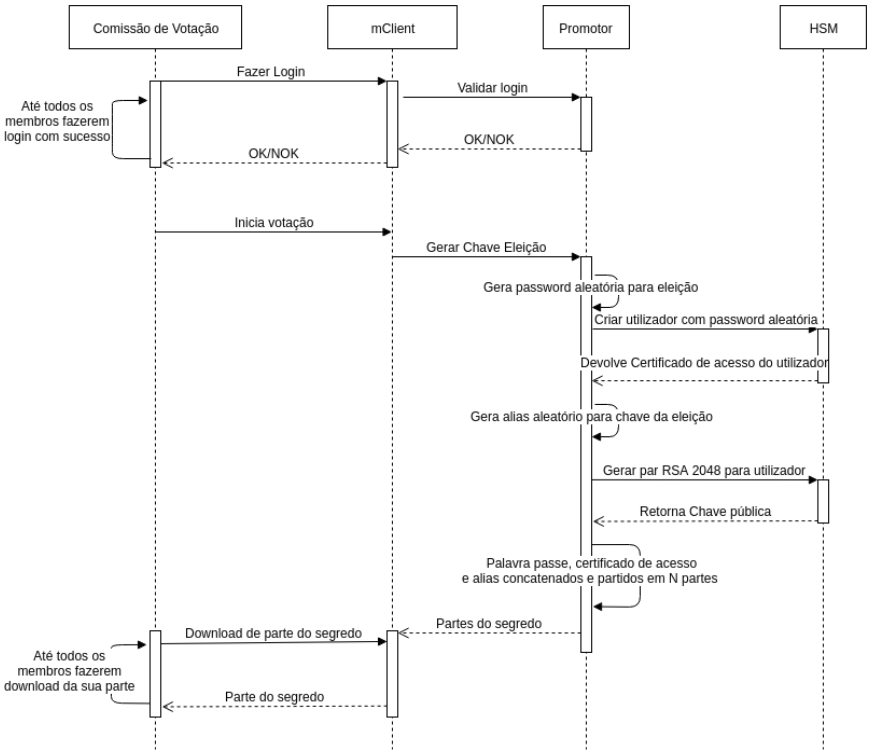
info@multicert.com

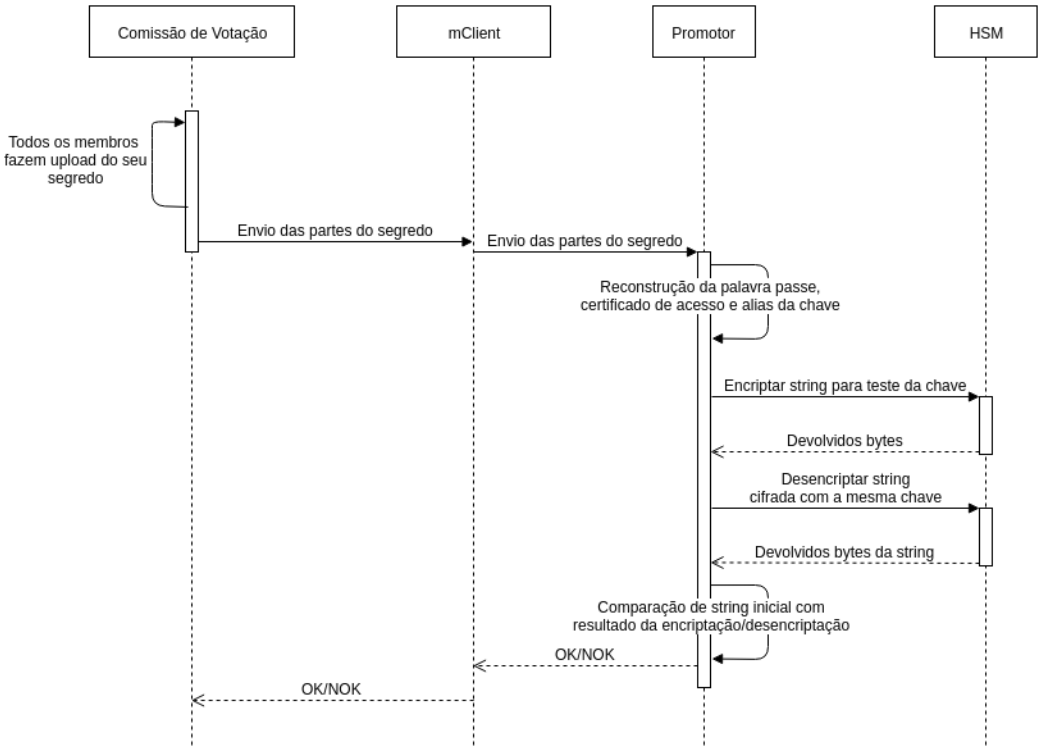
LISBOA
Lagoas Park, Edifício 3, Piso 3 - 2740-266 Porto Salvo – Oeiras - Portugal
Tel.: +351 217 123 010 | Fax: +351 217 123 011

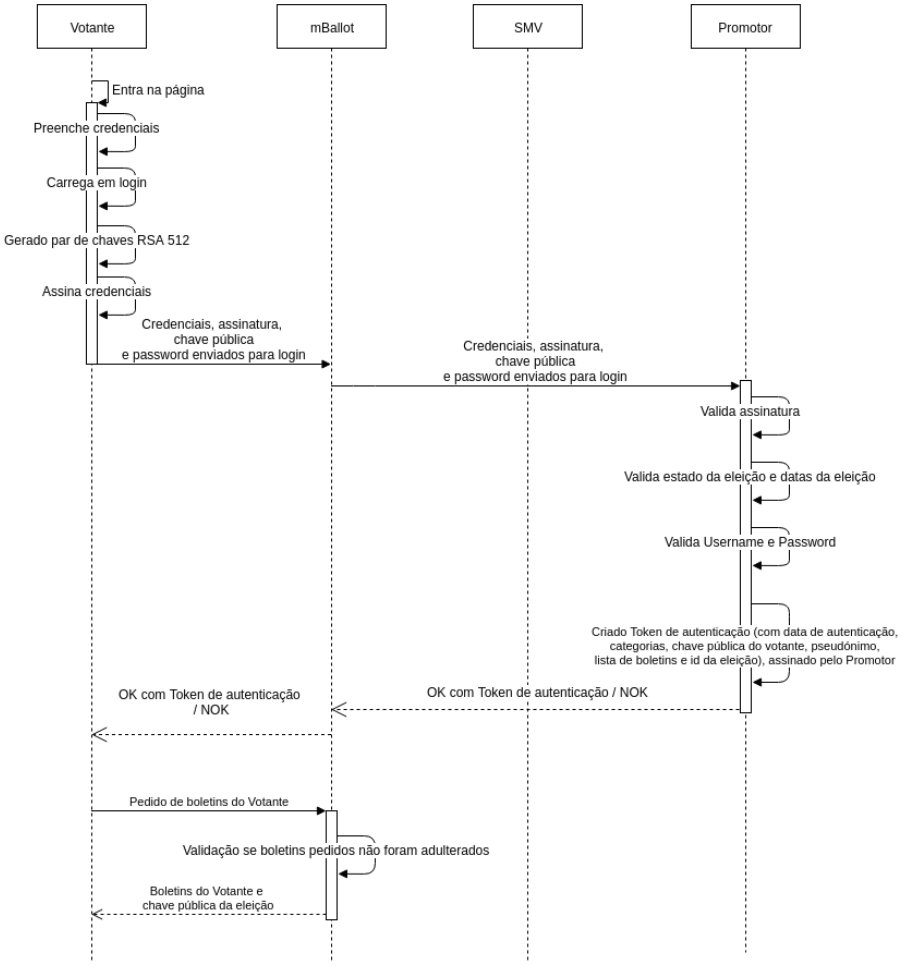
Appendix C

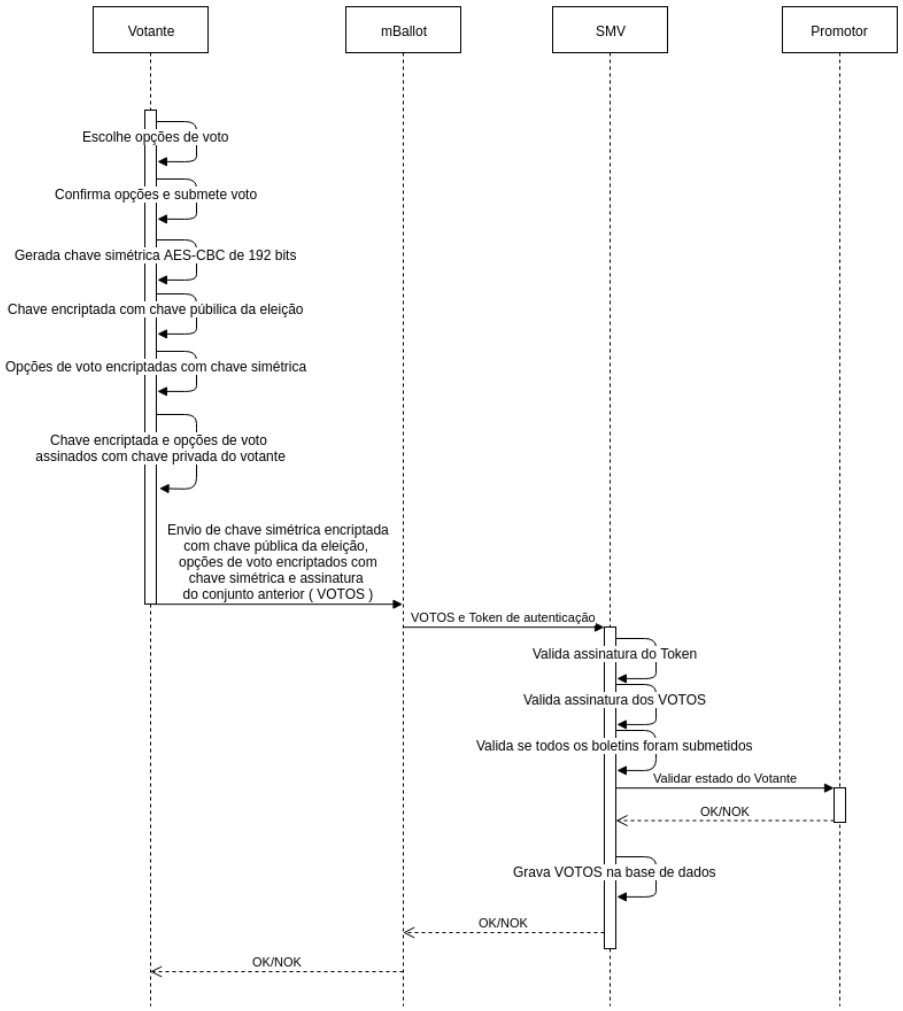
Certvote Diagram[1]

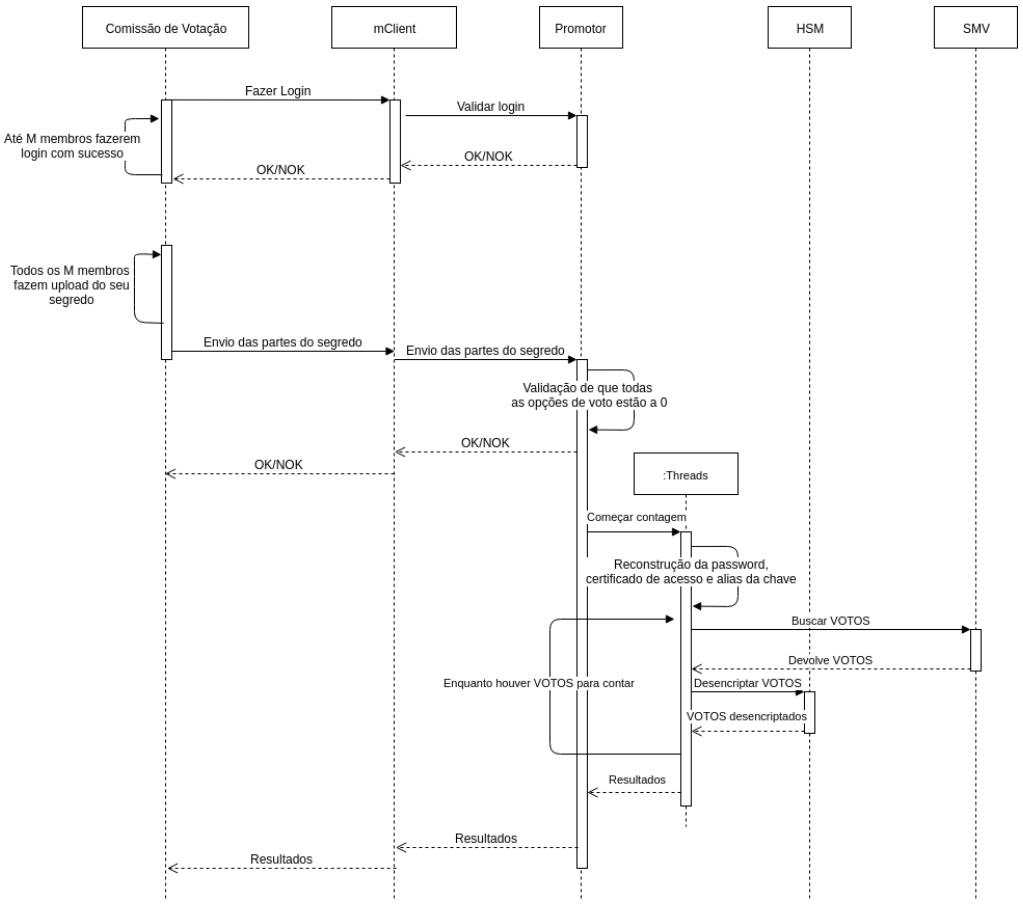
[1] As provided by Multicert to the author


Comissão de Votação
mClient
Promotor
HSM
Fazer Login
Validar login
Até todos os membros fazerem login com sucesso
OK/NOK
OK/NOK
Inicia votação
Gerar Chave Eleição
Gera password aleatória para eleição
Criar utilizador com password aleatória
Devolve Certificado de acesso do utilizador
Gera alias aleatório para chave da eleição
Gerar par RSA 2048 para utilizador
Retorna Chave pública
Palavra passe, certificado de acesso e alias concatenados e partidos em N partes
Partes do segredo
Download de parte do segredo
Até todos os membros fazerem download da sua parte
Parte do segredo


Comissão de Votação
mClient
Promotor
HSM
Todos os membros fazem upload do seu segredo
Envio das partes do segredo
Envio das partes do segredo
Reconstrução da palavra passe, certificado de acesso e alias da chave
Encriptar string para teste da chave
Devolvidos bytes
Desencriptar string cifrada com a mesma chave
Devolvidos bytes da string
Comparação de string inicial com resultado da encriptação/desencriptação
OK/NOK
OK/NOK


Votante
mBallot
SMV
Promotor
Entra na página
Preenche credenciais
Carrega em login
Gerado par de chaves RSA 512
Assina credenciais
Credenciais, assinatura,
chave pública
e password enviados para login
Credenciais, assinatura,
chave pública
e password enviados para login
Valida assinatura
Valida estado da eleição e datas da eleição
Valida Username e Password
Criado Token de autenticação (com data de autenticação,
categorias, chave pública do votante, pseudónimo,
lista de boletins e id da eleição), assinado pelo Promotor
OK com Token de autenticação / NOK
OK com Token de autenticação
/ NOK
Pedido de boletins do Votante
Validação se boletins pedidos não foram adulterados
Boletins do Votante e
chave pública da eleição


Votante
mBallot
SMV
Promotor
Escolhe opções de voto
Confirma opções e submete voto
Gerada chave simétrica AES-CBC de 192 bits
Chave encriptada com chave púbilica da eleição
Opções de voto encriptadas com chave simétrica
Chave encriptada e opções de voto assinados com chave privada do votante
Envio de chave simétrica encriptada com chave pública da eleição, opções de voto encriptados com chave simétrica e assinatura do conjunto anterior (VOTOS)
VOTOS e Token de autenticação
Valida assinatura do Token
Valida assinatura dos VOTOS
Valida se todos os boletins foram submetidos
Validar estado do Votante
OK/NOK
Grava VOTOS na base de dados
OK/NOK
OK/NOK


Comissão de Votação
mClient
Promotor
HSM
SMV
Fazer Login
Validar login
Até M membros fazerem login com sucesso
OK/NOK
OK/NOK
Todos os M membros fazem upload do seu segredo
Envio das partes do segredo
Envio das partes do segredo
Validação de que todas as opções de voto estão a 0
OK/NOK
OK/NOK
:Threads
Começar contagem
Reconstrução da password, certificado de acesso e alias da chave
Buscar VOTOS
Devolve VOTOS
Enquanto houver VOTOS para contar
Desencriptar VOTOS
VOTOS desencriptados
Resultados
Resultados
Resultados

Appendix D

Casa da Moeda Security[1]

[1] Source: https://www.incm.pt/portal/arquivo/matriz/200912.pdf, pages 8,9 - accessed January 2017

A INCM E A SEGURANÇA

Cofres na Casa da Moeda, para depósito de objectos pessoais.

Portas de segurança para acesso à área de produção da Unidade Gráfica.

A segurança é essencial para a confiança que o Estado, as instituições e os cidadãos depositam na INCM.

A INCM é reconhecida pelos seus parceiros pela sua imagem de confiança, indispensável ao papel relevante que desempenha na sociedade, como fabricante de moeda, como produtor e personalizador de documentos de identidade e de viagem e de outros documentos seguros, indispensáveis no mundo actual, caracterizado pela mobilidade e globalização.

A segurança é essencial para a confiança que o Estado, as instituições e os cidadãos depositam na INCM. E, como todos sabemos: «A confiança chega a pé devagar e parte a cavalo a galope.»

As questões da segurança são decisivas nos produtos, nas instalações, nos sistemas lógicos e nos processos de gestão, personalização e controlo, para além do ambiente geral de segurança e da atitude responsável de cada um.

Segurança comportamental

O atributo de segurança de uma instituição é conseguido, em primeiro lugar, pelas qualidades das pessoas que nela trabalham, pela consciência das razões e objectivos dessa segurança e por uma disciplina de actuação apoiada em procedimentos claros e exaustivos que, para serem efectivos, têm de ser permanentemente revistos, actualizados e recordados.

A atitude de segurança, que não significa espírito securitário mas, antes, de cuidado e atenção pelo cumprimento rigoroso dos procedimentos, designadamente em situações inesperadas ou excepcionais, é decisiva para a confiança recebida de diversas entidades nacionais e estrangeiras e que está traduzida nas licenças e certificações obtidas e na encomenda dos produtos que fabricamos.

Segurança física

A INCM toma todas as medidas adequadas para proteger instalações, equipamentos e materiais, contra danos acidentais, acessos não autorizados e perigos de origem criminosa:

- Instalações de alta segurança, com sistemas de controlo;
- Acesso dos materiais distinto do acesso das pessoas;
- Rastreabilidade de movimentos e operações;
- Destruição controlada dos materiais;
- Cofres blindados, com acesso restrito e condicionado;
- Equipamentos de controlo de acesso e movimentos;

Segurança lógica

A INCM impede o acesso lógico a bens e dados lógicos por meios não físicos, protegendo sistemas, aplicações e dados:

- Isolamento das instalações onde estejam equipamentos com dados e aplicações sensíveis, *drives* e conexões activas;
- Colocação de bases de dados e aplicações informáticas sensíveis numa LAN interna (*local area network*) sem ligação a outras redes;

Leitores para abertura das portas, com banda magnética e dados biométricos, e portas de segurança de acesso à zona de alta segurança.

·Autenticação de utilizadores e equipamentos;
·Impossibilidade de acesso individual à administração de sistemas;
·Selagem das cablagens;
·Não partilha dos equipamentos por aplicações não ligadas ao processo;
·Restrições no acesso lógico ao *software* e dados.

Segurança dos dados

A INCM garante a confidencialidade e integridade dos dados pessoais e sensíveis referentes à personalização e emissão de documentos:
·Comunicações externas por redes seguras;
·Registo exaustivo de dados para auditoria;
·Processos criptográficos, emissão de chaves e outras operações exclusivamente através de HSM (*hardware security modules*);
·Procedimentos de rotina seguros para a realização de cópias de segurança de dados e sistemas.

Continuidade das operações

A INCM assegura a continuidade permanente das operações, definindo planos de continuidade e de contingência com as entidades emissoras responsáveis:
·Planos de continuidade, com identificação de factores de risco ou de falha, naturais, de equipamento e humanos, e a forma de os evitar ou reduzir, através da combinação de controlo preventivo e de recuperação;
·Planos de contingência que possibilitem a continuidade dos processos nos limites de tempo e condições exigíveis.**M**

References

[1] Morris Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, DTIC Document, 2001. URL: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA400014.

[2] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981. URL: http://dl.acm.org/citation.cfm?id=358563.

[3] Ben Adida. Helios: Web-based Open-Audit Voting. In *USENIX Security Symposium*, volume 17, pages 335–348, 2008. URL: https://www.usenix.org/legacy/event/sec08/tech/full_papers/adida/adida.pdf.

[4] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *International Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251. Springer, 1992. URL: https://people.csail.mit.edu/rivest/voting/papers/FujiokaOkamotoOhta-APracticalSecretVotingSchemeForLargeScaleElections.pdf.

[5] David Bernhard and Bogdan Warinschi. Cryptographic Voting—A Gentle Introduction. In *Foundations of Security Analysis and Design VII*, pages 167–211. Springer, 2014. URL: https://eprint.iacr.org/2016/765.

[6] Manon de Vries and Wouter Bokslag. Evaluating e-voting: theory and practice. *CoRR*, (arXiv:1602.02509v), 2016. URL: https://arxiv.org/pdf/1602.02509.pdf.

[7] Syed Taha Ali and Judy Murray. An Overview of End-to-End Verifiable Voting Systems. *arXiv preprint arXiv:1605.08554*, 2016. URL: http://arxiv.org/abs/1605.08554.

[8] Peter Hyun-Jeen Lee and Siamak F. Shahandashti. Theoretical Attacks on E2e Voting Systems. Technical Report 447, 2016. URL: http://eprint.iacr.org/2016/447.

[9] Olivier Pereira. Internet Voting with Helios. In *Real-World Electronic Voting: Design, Analysis and Deployment*, page 461. CRC Press, 2016. URL: http://www.uclouvain.be/crypto/services/download/publications.pdf.9f0ecdd41f6c72bc.636831312e706466.pdf.

[10] Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 10–18. Springer Berlin Heidelberg, August 1984. URL: http://caislab.kaist.ac.kr/lecture/2010/spring/cs548/basic/B02.pdf.

[11] William Stallings. *Cryptography and network security: principles and practices*. Prentice-Hall, Inc, 2006.

[12] Jakob Jonsson and Burt Kaliski. Public-key cryptography standards (PKCS)# 1: RSA cryptography specifications version 2.1. 2002. URL: http://tools.ietf.org/html/rfc3447).

[13] NIST FIPS Pub. 197: Advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197:441–0311, 2001.

[14] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979. URL: http://dl.acm.org/citation.cfm?id=359176.

[15] J Falcão et al. Auditing e-voting pilot processes and systems at the elections for the european parliament and for the portuguese parliament. *E-voting: The last electoral revolution*, pages 93–114, 2008.

[16] Nigel P Smart, European Union, and European Network and Information Security Agency. *Algorithms, key size and parameters: report - 2014*. ENISA, Heraklion, 2014. OCLC: 903500575. URL: http://bookshop.europa.eu/uri?target=EUB:NOTICE:TP0514084:EN:HTML.

[17] Ralf Kusters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. In *2012 IEEE Symposium on Security and Privacy*, pages 395–409. IEEE, 2012. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6234426.

[18] Ryan W. Gardner, Sujata Garera, and Aviel D. Rubin. Coercion resistant end-to-end voting. In *International Conference on Financial Cryptography and Data Security*, pages 344–361. Springer, 2009. URL: http://link.springer.com/10.1007%2F978-3-642-03549-4_21.

[19] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachene. Election verifiability for Helios under weaker trust assumptions. In *Computer Security-ESORICS 2014*, pages 327–344. Springer International Publishing, 2014.

[20] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 626–643. Springer, 2012. URL: http://link.springer.com/chapter/10.1007/978-3-642-34961-4_38.