UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

A Fault Tolerant Voting System for the Internet

Rui Filipe Lopes Joaquim
(Licenciado)

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Orientador: Doutor André Ventura da Cruz Marnôto Zúquete
Co-Orientador: Doutor Paulo Jorge Pires Ferreira

Júri

Presidente: Doutor Paulo Jorge Pires Ferreira

Vogais: Doutor André Ventura da Cruz Marnôto Zúquete
Doutor Manuel Bernardo Martins Barbosa
Doutor Fernando Henrique Corte Real Mira da Silva

Fevereiro de 2005

Tese realizada sob orientação do

Doutor André Ventura da Cruz Marnôto Zúquete

Professor Auxiliar do Departamento de Electrónica e Telecomunicações da

Universidade de Aveiro

e co-orientação do

Doutor Paulo Jorge Pires Ferreira

Professor Associado do Departamento de Engenharia Informática e de

Computadores do Instituto Superior Técnico.

Resumo

Vários países estão a efectuar reformas nos seus processos eleitorais de forma a substituir os velhos sistemas de voto tradicional baseados em papel. Com a generalização do acesso à Internet, a possibilidade de votar usando um sistema de voto pela Internet passou de um sonho a uma possibilidade. O voto pela Internet é alvo de estudo à mais de duas décadas por vários investigadores. Vários protocolos foram propostos e alguns deles implementados. No entanto, os vários protótipos desenvolvidos dão apenas suporte às interacções básicas do protocolo e não fornecem mecanismos de suporte a falhas, muito embora estes sejam críticos para uma utilização real do sistema.

Nesta dissertação é apresentado o sistema de voto REVS que apresenta as características fundamentais dos sistemas de voto: democracia, privacidade, correcção e verificabilidade. No entanto, o principal objectivo do REVS é providenciar mecanismos capazes de tolerar falhas em máquinas e/ou comunicações que possam levar à interrupção do protocolo de votação. Assim, o sistema REVS foi concebido para permitir: (i) interromper e reatar o processo de votação sem enfraquecer o protocolo de voto; (ii) suportar falhas nos servidores através de replicação; e (iii) impedir que um só servidor, sozinho ou até um certo nível de conluio, consiga corromper o resultado da eleição.

Palavras-chave: Voto pela Internet, Voto Electrónico, Tolerância a Falhas, Escalabilidade, Facilitação

Abstract

Today many countries are conducting reforms in electoral system for replacing old paper-based voting systems by electronic voting systems. With the explosive growth and consequent usage of the Internet the possibility of using it for voting has became real. Researchers have been working on electronic voting issues for more than two decades and many protocols have been proposed. However, only a few protocols have prototypes implemented, and usually those prototypes are focused in supporting the basic protocol interactions and do not handle properly some real world issues, such as fault tolerance.

In this thesis we start with an overview of electronic voting, highlighting the techniques used in the various proposed protocols. We also analyze the voting protocols and point-out the pros and cons of them. Then we present REVS, a robust electronic voting system designed for distributed and faulty environments, namely the Internet. REVS is an electronic voting system that accomplishes the desired characteristics of traditional voting systems, such as accuracy, democracy, privacy and verifiability. But the main goal of REVS is deal with failures in real world scenarios, such as machine or communication failures, which can lead to protocol interruptions. REVS robustness has consequences at three levels: (i) the voting process can be interrupted and recovered without weakening the voting protocol; (ii) it allows a certain degree of failures with server replication; and (iii) none of the servers conducting the election, by its own or to a certain level of collusion, can corrupt the election outcome.

Keywords: Internet Voting, Electronic Voting, Fail Tolerance, Scalability, Facilitation

À minha esposa

Rosa Henriques

Aos meus pais

Vicente Maria Joaquim

Olívia Lopes Pires

Agradecimentos

Ao Prof. Doutor André Zúquete pela exemplar orientação ao longo de todo o trabalho realizado. Agradeço também pelas valiosas discussões e revisões que contribuíram de forma decisiva para esta tese.

Ao Prof. Doutor Paulo Ferreira pela orientação, ideias, revisões e discussões ao longo de todo o trabalho realizado.

A todos os membros do grupo de Sistemas Distribuídos do INESC-ID por todo o seu apoio.

Por fim, mas não por último, um agradecimento muito especial à minha família e amigos que sempre me apoiaram e incentivaram.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last few years several experiments have been conducted in order to facilitate elections. The facilitations were introduced by new ways of expressing votes besides the traditional paper-based. Examples of new voting interfaces and systems are touch screens, SMS messages from cellular phones and distributed voting systems using the Internet [MSOA01, UKD].

Motivation

Internet voting systems are appealing for several reasons: (i) people are getting more used to work with computers to do all sort of things, namely sensitive operations such as shopping and home banking; (ii) they allow people to vote far from where they usually live, thus helping to reduce abstention rates; and (iii) they may support arbitrary voting ballots and check their correct fulfillment during the voting process.

Problems

Internet voting systems face several problems that prevent their widespread use today [Cal00, Cal01, Cra01, Int01, Riv01, Rub02]. The problems can be

broadly divided in three main classes.

The first class includes security and fault tolerance problems inherited from the current Internet architecture. Vital services, such as DNS name resolution, can be tampered in order to mislead users into spoofing servers [LMMM00]. IP routing mechanisms and protocols, managed by many different organizations, should deal with partial communication outages; however, communication problems may still arise.

The second class includes problems that are specific to voting protocols. These problems derive from the assumptions of the protocols about the execution environment, namely:

- Client machines used by voters must be trusted, in order to act as trusted agents, which is hard to ensure in personal or multi-user computers with general-purpose commercial operation systems.
- Servers controlling the voting process do not (i) fail, (ii) become unreachable or (iii) pervert the voting protocol. The protocol perversion includes either not reacting properly to client requests or by trying to influence the election by acting as a voter.
- The voting protocol is not disturbed by communication problems or machine failures.

The third class includes problems that may be created by specific attacks against a voting protocol or a running election. Such attacks may try to get some useful outcome, by subverting the voting protocol, or simply ruin an election using Denial of Service (DoS) attacks against the participating machines or applications. Another kind of attack is the coercion of voters, which can happen if they can vote anywhere without supervision of electoral committees or other trustworthy agents.

Contribution

REVS is an Internet voting system that was designed to tackle some of these problems. In particular, the REVS voting protocol, involving several participating machines, supports some types of communication and machine failures by keeping a distributed loosely-coupled state. Each voter keeps a local state, in mobile non-volatile storage, allowing him to stop and resume the voting process anytime and anywhere. Servers are replicated and only a subset of them needs to be contacted by each voter. Each server keeps a distinct state regarding the participation of each voter in the election, and allows voters to get many times the same answer from each server. Each server alone is not able to act as any voter and cannot provide false replies to voters without being noticed. The collusion of servers in order to interfere with the election (e.g. voting for absentees) is prevented to a certain degree of collusion.

Technically REVS is a blind signature electronic voting system based in Evox and Evox Managed Administrators (Evox-MA) systems. The Evox-MA was proposed by DuRette in 1999 [DuR99], and improves the 1997 Herschberg's Evox system [Her97]. The Evox-MA improvements over Evox were to eliminate single entities capable of corrupting the election. Both Evox and Evox-MA systems are very sensible to failures in communication or servers, a problem that we solved with REVS. Furthermore, the Evox-MA system has problems concerning the authentication of voters, allowing an easy impersonation of voters by the servers running the election. In REVS we solved this problem redesigning the voter's authentication algorithm.

Electronic voting is a very delicate process that needs to be trusted by the public. For allowing a public evaluation of electoral systems it is advised to use an open-source approach, thus REVS is open-source (available at

www.gsd.inesc-id/~rjoaquim). With open-source software the system may be analyzed by many people, therefore it is easier to detect any bugs and to correct them.

Experiment

A first prototype of REVS was deployed at Instituto Superior Técnico to support elections, namely pedagogic quality surveys. To this particular scenario, REVS servers were deployed and managed by separate entities, namely central computer services, several departments and student's organizations, in order to reduce the possibility of collusion. A set of trusted machines was made available to voters, though they could use any machine to participate in the elections. The voters used a Voter's Module that is a signed Java program. The Voter's Module was able to check the correct fulfillment of ballots and to contact the correct REVS servers to submit them.

The structure of this document is the following: Chapter 2 presents the phases and properties of electronic voting systems and also the issues related to the deployment of them; Chapter 3 presents the main cryptographic mechanisms used by electronic voting protocols; Chapter 4 resumes the work that has been done on the field; Chapter 5 presents REVS; Finally, in Chapter 6, we draw some conclusions and also point out to some future work.

Chapter 2

Understanding an electronic voting system

In this Chapter we will provide the essential information to understand electronic voting systems. We start by presenting the usual phases of a voting process. Then we describe the properties required for an electronic voting system. Finally we look at the issues concerning the deployment of such a system.

2.1 Voting phases

The phases of an electronic voting process are similar to the phases of a traditional paper-based voting process, which are the following:

- Registration: The registration phase consists in compiling the list of the eligible voters and to provide them some credentials needed to apply in future elections.
- Validation: The validation phase consists in verifying the credentials of the voters during an election. Only registered voters with the proper

credentials may be authorized to vote, and only once per election.

- Collection: The collection phase consists in collecting the votes from all the participating voters.

- Verification: In this phase it is verified the validity of the ballots. Only the valid ballots are used in the tallying phase.

- Tallying: The tallying phase consists in counting votes from the valid ballots. At the end the tally is published.

- Claiming: This is the phase in which all the claims should be made and investigated. The claiming phase should run in parallel with all the other phases. At the end of this phase the final results are published.

2.2 Voting properties

Researchers in the electronic voting field have already reached a consensus pack of four core properties that an electronic voting system should have [CC97]:

- **Accuracy:** (1) it is not possible for a vote to be altered, (2) it is not possible for a validated vote to be eliminated from the final tally, and (3) it is not possible for an invalid vote to be counted in the final tally.

- **Democracy:** (1) it permits only eligible voters to vote, (2) it ensures that eligible voters vote only once, and (3) ensures the equality of knowledge, i.e. no partial results.

- **Privacy:** (1) neither authorities nor anyone else can link any ballot to the voter who cast it and (2) no voter can prove that he voted in a particular way.

The second part of this property exists to assure that no one can coerce or bribe a voter. The systems with this second part of the Privacy property are usually called Receipt-free voting systems.

- **Verifiability:** anyone can independently verify that all votes have been counted correctly.

Accuracy, democracy and verifiability are, in most cases of today's electoral systems, assured by the presence of representatives of opposite parties. The privacy property is currently assured by the existence of private voting booths controlled by electoral committees, allowing voters to cast their votes in secrecy and without coercion.

The presented four core properties are essential properties that voting systems should have, but they do not deal with operational requirements, which are:

- **Availability:** (1) the system works properly as long as the poll stands and (2) any voter can have access to it from the beginning to the end of the poll.

- **Resume-ability:** the system allows any voter who had interrupted his voting process to resume or restart it while the poll stands.

- **Collusion-resistance:** the collusion-resistance measures the capacity of the system to resist undisturbed when misbehaviour from any electoral entity or group of entities occurs.

  If all supervising entities conspire this property isn't achieved. So, this characteristic should be measured in terms of the total number of entities that must conspire to guarantee a successful interference in the election.

We say that a system is Robust if he holds the last tree properties described.

2.3 Electronic voting deployment

There are two broad categories of Internet electronic voting systems that must be distinguished in any discussion about Internet voting [Cal00]. The difference is based on whether or not the election agency has full control over the client-side infrastructure and software used for voting:

- Agency-controlled systems: In these systems the actual computers and software used for voting, along with the networks to which they are immediately attached, and the physical environment of voting, are under the control of election officials (or their contractors, etc.) at all times.

- Vote-from-anywhere systems: These are systems intended to support voting from essentially any computer connected to the Internet anywhere in the world, e.g. from home, workplaces, schools, hotels, cybercafés, military installations, handheld appliances, etc. In this case the computers used as voting machines, the software on them, the networks where they are immediately attached to and the physical surroundings are under the control of the voter or of a third party, but not under the control of election officials.

This distinction is fundamental because with systems that are not agency-controlled, the voting environment is difficult to secure against privacy hazards and security attacks that can arise from infection with malicious code or use of remote control software. Hence, white vote-from-anywhere systems it should be substantially more complex to achieve the same degrees of privacy and security that are achievable with agency-controlled systems.

2.3.1 Four-stage approach to implement Internet Voting

We defend a incremental approach in the deployment of Internet voting systems, such as the one presented by the California Internet Voting Task Force [Cal00]. This task force presented a four-stage approach for introducing secure Internet voting systems. Each stage is a technical advance on the previous ones, but provides better service to more voters. These four stages are:

1. Internet voting at voters' precinct polling place: Internet-connected computers are deployed at regular precinct polling places alongside traditional voting systems on election day. Voters identify themselves to clerks as usually with the traditional system, and then have their choice of voting methods.

2. Internet voting at any polling place: Systems of this type are similar to (1), except that the voter need not show up on the election day at his own precinct polling place, but may vote at any precinct polling place equipped for Internet voting, or at any other polling places that might be set up at shopping centers, schools, or other places convenient to voters. Non-precinct polling places might be open for early voting for days or weeks in advance of election day, possibly with extended hours. Such sites would still be manned by agency personnel, but they would have to have access to the entire voter roll to check registration and prevent duplicate voting, rather than just the roll for one precinct. This might itself be implemented with Internet access to the voter's registration database.

3. Remote Internet voting at agency-controlled computers or kiosks: Systems of this type are similar to (2) except that the polling places should

not have to be manned by trained county personnel, but only be responsible lower-level clerks whose job is to safeguard the voting computers from tampering, restart them when necessary, and call for help if needed.

4. Remote Internet voting from home, office, or any Internet-connected computer: These systems permit voting from essentially any Internet-connected PC, anywhere, including home, office, school, hotel, etc.. At voting time, the voters must first secure the computer against malicious code and remote control software somehow, then connect to the proper voting site, authenticate themselves, retrieve an image of the proper ballot, and vote.

The first three of these system types are agency-controlled systems, as defined in Section 2.3. We believe that these systems can reasonably be deployed, at least for trial purposes, as soon as they are built and certified as satisfying all the requirements.

The last type of system is of the category of vote-from-anywhere systems, also described in Section 2.3. The deployment of these systems is not recommended until a satisfactory solution to the malicious code and remote control software problems is offered.

Chapter 3

Electronic voting concepts and techniques

In this Chapter we try to give a simple and clear idea of the basic concepts and techniques behind the design and implementation of many electronic voting protocols. For a wider introduction to cryptographic concepts and mechanisms see [MOV97].

3.1 Public key encryption

In the public key encryption, also known as asymmetric encryption, there are two keys: an encryption key *Kpub* (public key) and a decryption key *Kpri* (private key). The encryption of a message m with *Kpub* results in c, to recover m from c we use *Kpri*, as follows:

$$c = E_{Kpub}(m)$$

$$m = D_{Kpri}(c) = D_{Kpri}(E_{Kpub}(m))$$

In electronic voting a public key cryptosystem is normally used to provide secure authentication to the voters, see Section 3.3, or to establish secure connections between the voters and the electoral servers.

3.1.1 RSA public key cryptosystem

The most known and used algorithm for public key encryption is the RSA, proposed by Rivest, Shamir and Adleman in 1977 [RSA77]. The security of the RSA algorithm is based on the problems of factorization and calculation of modular logarithm for large numbers. In electronic voting the use of the RSA, or some derived algorithms (see Section 3.3), is common on blind signature based voting systems (see Section 3.3.1). It is also used in the construction some of mix-nets (see Section 3.6). The details of the algorithm are shown in Table 3.1.

Secret values	p, q	Secret distinct large primes, also calculate $\varphi = (p-1)(q-1)$
Public value	n	$n = p.q$
Public key	e	$1 < e < n$, such that $\gcd(e, \varphi) = 1$
Private key	p, q, d	$d < n$, such that $1 = e.d \bmod \varphi$
Encryption	$c = m^e \bmod n$	
Decryption	$m = c^d \bmod n$	

Table 3.1: RSA algorithm

3.1.2 ElGamal public key cryptosystem

The ElGamal public key cryptosystem was proposed in 1985 [ElG85] by Taher ElGamal. The security of the algorithm is based on the problem of the calculation of modular logarithm for large numbers. Due to some

Public values	p, g	p is a large prime number and g is a generator of $\mathbb{Z}^*_p$
Private key	a	a is some random number such that $a \in \mathbb{Z}_{p-1}$
Public key	(p,g,γ)	$\gamma = g^a \bmod p$
Encryption	$c = (y1, y2)$	$r < p - 1$ is a secret random number. $y1 = g^r \bmod p$ $y2 = \gamma^r.m \bmod p$
Decryption	$m = y2.y1^{-a} \bmod p$	

Table 3.2: ElGamal algorithm

characteristics of ElGamal cryptosystem it is used in homomorphic based voting systems (see Section 4.1.5), and in some of the mix-net based voting systems (see Section 3.6). The details of the algorithm are shown in Table 3.2.

The ElGamal public key cryptosystem has some very useful properties for electronic voting. A brief explanation of those properties follows [BT94, HS00, MOV97, Mür00]:

Random re-encryptability: It's possible to re-encrypt a ciphertext y, of the message x, producing another ciphertext y' of the same message x without knowing the x. In the case of ElGamal we can re-encrypt a cryptogram y

$$y = E(x) = (g^r, \gamma^r x) = (y_1, y_2)$$

by multiplying the first component by $g^{r'}$ and the second by $\gamma^{r'}$, where $r' \in \mathbb{Z}_{p-1}$is a newly chosen random number:

$$y' = (g^{r'} y_1, \gamma^{r'} y_2) = (g^{r+r'}, \gamma^{r+r'} x)$$

A cryptosystem with this property can be used in the construction of mix-nets, see Section 3.6.

Homomorphism property: For some cryptographic systems there exist some polynomial-time computable functions $\otimes, \oslash, \oplus, \ominus$, that for any messages x_1, x_2 hold two properties:

1. The additive homomorphic property

$$E(x_1) \otimes E(x_2) = E(x_1 \oplus x_2) \tag{3.1}$$

2. The subtractive homomorphic property

$$E(x_1) \oslash E(x_2) = E(x_1 \ominus x_2) \tag{3.2}$$

In the case of ElGamal cryptosystem $\oplus$ and $\ominus$ are, respectively, the multiplication and division in $\mathbb{Z}^*{}_p$. The operation $\otimes$ is component-wise multiplication and $\oslash$ is component-wise division.

The most interesting property for electronic voting is the additive homomorphic property. Namely it allows to decipher the overall tally instead of each individual vote in homomorphic voting systems, Section 4.1.5.

Provable decryption: Having a ciphertext c and the decrypted plain text $x = D(c)$, it's possible to prove that the decryption operation succeed without revealing anything about the private key. With ElGamal it's enough to prove that

$$\log_g \gamma = \log_{y1}(y_2/x)$$

This can be accomplished with the help of zero knowledge proofs (see Section

Secret values	p, q	Secret distinct large primes.
Public value	n	$n = p.q$
Public key	g	Where g is an integer of order a multiple of n mod n^2
Private key	$\lambda(n)$	$\lambda(n) = \text{lcm}((p-1)(q-1))$
Encryption	$c = g^m . r^n \bmod n^2$	
Decryption	$m = (L(c^{\lambda(n)} \bmod n^2) / L(g^{\lambda(n)} \bmod n^2)) \bmod n$, where $L(u) = u - 1 / n$.	

Table 3.3: Paillier's algorithm

3.4). This is very important in the case of homomorphic voting systems, because if the private key need to be made public to prove that the decryption of the overall tally is correct it would be possible to decipher each individual, therefore breaking the voters anonymity, see Section 4.1.5.

3.1.3 Paillier's public key cryptosystem

The Paillier cryptosystem was proposed by Pascal Paillier in 1999 [Pai99]. Its security is based on the composite residuosity class problem. This cryptosystem has the additive homomorphic property (Equation 3.1), so it is useful for homomorphic based voting systems, see Section 4.1.5. The details of the algorithm are shown in Table 3.3.

3.2 Secret sharing

Secret sharing, as the name suggests, is called to the process of sharing a secret S among N parties so that only t or more parties can later recreate the secret. Each party P_i keeps his share s_i secret, so that just $m \geq t$ parties

can recreate the secret S. Such a scheme it's called (t, N)-*threshold secret sharing scheme.* The interest of this scheme is to prevent the ability of less than t parties to reveal the shared secret.

3.2.1 Threshold cryptosystem

In a threshold cryptosystem the secret sharing technique is used to share a private key *Kpri* among N parties, in such a way that at least t parties must cooperate to decrypt $E_{Kpub}(m)$, where m is an arbitrary message. These systems are called (t,N)-*threshold cryptosystem*s. Threshold cryptosystems usually include two algorithms:

Key generation protocol: All the N parties are involved in the generation of the share public key *Kpri*. At the end each one receives its share of the private key *Kpri*.

Verifiable decryption protocol: Allows t parties to cooperatively decrypt an encrypted message $E_{Kpub}(m)$ in a way that everyone can verify that the decryption was performed correctly. This process should not give anyone the ability to decrypt alone any other messages encrypted with the same public key.

In some electronic voting protocols there is an election public key, used to encrypt the ballots. The use of a *threshold cryptosystem* for the election's private key brings obvious improvements to the system security, because votes cannot be revealed without the cooperation of t election authorities.

3.3 Digital signatures

Digital signatures are the digital equivalent of hand-made signatures. They are normally produced using a public key cryptosystem, or one variation of

it, where the private key *Kpri* is used to sign a message m and the public key *Kpub* is used to verify the signature. The most popular digital signature algorithms are the RSA and the DSA (Digital Signature Algorithm, a standard that is a variation of the ElGamal encryption algorithm). The algorithm of the RSA digital signatures is similar to the RSA public key encryption, but in this case we use the private key for encryption (signing) and the public key for decryption (signature verification), as follows:

Signing: $s = m^d \bmod n$

Verification: $m = s^e \bmod n$

Note that, due to performance constrains, normally the digital signature is produced over a digest of the message, instead of the full message.

In electronic voting digital signatures are normally used to authenticate voters, election data and voting protocol messages.

3.3.1 Blind signatures

Blind signatures are a class of digital signatures proposed in 1982 by David Chaum [Cha82]. The goal of blind signatures is to allow an entity A to obtain a signature of another entity B on a message m without revealing m to B. Making a parallel with the paper-based world, getting a blind signature is similar to setting a signature over a document bellow a sheet of carbon inside an envelope; if someone signs the outside of the envelope, then also signs the document inside the envelope. The signature remains attached to the document, after being removed from the envelope. This construction is used on the blind signature based voting systems to validate ballots (cf. Section 4.1.3). The algorithm for creating blind signatures based on RSA signatures is as follows:

Blinding: $m' = r^e.m \bmod n$, where r is a random number $< n$ (public key)

Signing: $bs = m'^d \bmod n$ (private key)

Unblinding: get r^{-1} such that $r.r^{-1} \equiv 1 (\bmod n)$ (public key) then:

$$s = r^{-1}.bs \equiv r^{-1}.(r^e.m)^d \equiv r^{-1}.r^{ed}.m^d \equiv r^{-1}.r.m^d \equiv m^d (\bmod n)$$

3.4 Zero-knowledge proofs

Sometimes we need to prove the knowledge of information without revealing anything about it; this is called a zero-knowledge proof. Zero-knowledge proofs were introduced in 1985 by Goldwasser, Micali, and Rackoff [GMR85]. They are probabilistic proofs that demonstrate membership in the language without conveying any additional knowledge, in such a way that a verifier achieves an desirable level of certain. For more detailed information see [Gol95]. It is usual to use zero-knowledge proofs in electronic voting to prove the correctness of mix-nets (see Section 3.6) and the correctness of the tally decryption in homomorphic voting systems (see Section 4.1.5).

3.5 Bulletin Board

Some cryptographic protocols, to prove their correctness, need a space where everyone can write and read but not delete information. A Bulletin Board (BB) is the name given to some kind of repository with all the previous mentioned properties (everyone can write/read, cannot delete).

Due to the distributed nature of Internet voting there must be taken some measurements concerning the availability of the BB. In the case of electronic voting all writes in the BB must be authenticated to prevent frauds. If all

messages are authenticated the BB are very useful for auditing proposes.

3.6 Mix-Net

The mix-net concept was introduced in 1981 by David Chaum [Cha81]. The primary goal of a mix-net is to provide anonymity in communications; therefore it is a natural construction block to achieve anonymity in electronic voting systems, normally when submitting votes.

The building part of a mix-net is the mix server. Each mix server has the role of hiding the correspondence between its input and output. There are two main approaches to build mix-nets: the encryption approach and the re-encryption approach.

Encryption approach: In the encryption approach there is an asymmetric key pair for every mix server and works as follows:

1. There are N mix servers $M_1, ..., M_N$, each one with an asymmetric key pair (K_i^+, K_i^-).

2. An entity who wants to send a message m anonymously encrypts it successively with the public key of the mixes. A random factor r_i is added in each encryption to avoid an easy correlation between the input and the output of each mix server

$$E_{K_1^+}(E_{K_2^+}(...E_{K_N^+}(m, r_N), r_2), r_1)$$

3. Then the entity sends the result to the first mix, M_1, who removes the first encryption with his private key K_1^-. Then, it throws away the random factor r_1, permutes the decrypted messages received and

sends them to the next mix server.

$$D_{K_1^-}(E_{K_1^+}(E_{K_2^+}(...E_{K_N^+}(m, r_N), r_2), r_1)) = E_{K_2^+}(...E_{K_N^+}(m, r_N), r_2)$$

4. The last mix server removes the last layer of encryption an delivers the message m.

By throwing away random factors after each decryption and shuffling the messages before forwarding them to the next mix server, the relation between the input of the first mix server and the output of the last one is perfectly hidden. The main drawback of this approach is that it requires the availability of all mixes, which is a strong assumption for faulty environments.

Re-encryption approach: A re-encryption mix-net uses the re-encryption property of some public key cryptosystems such as ElGamal, c.f. Section 3.1.2. In this approach a mix server re-encrypts a message m and permutes it with the other messages, then delivers the result to the next mix server, see Figure 3.1.

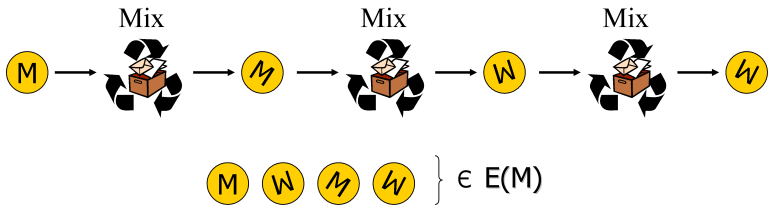

Figure 3.1: Re-encryption mix-net

At the end the re-encrypted message is decrypted with a (t,N)-*threshold* algorithm. This approach requires a (t,N)-*threshold* public key cryptosystem

with the re-encryption property. But it provides a simpler method to use and also a smaller ballot length due to the constant ciphertext length. In the encryption mix-net there is a fixed order to decrypt the message though out all mixes, while in the re-encryption approach the mixing can be done randomly and any t entities sharing the private key can decrypt the message at the end.

Both approaches require a proof that each mix performed well. This is usually achieved with zero knowledge proofs (cf. Section 3.4), but those can be very expensive in terms of size and computation requirements.
A solution to this problem, the Randomized Partial Checking (RPC), was presented in 2002 by Jacobsson, Juels and Rivest [JJR02] and fits either approach. They propose that a mix server should reveal the connection between some random selected inputs and the corresponding outputs; if the verification between the input and output succeeds then the verifier achieves a level of certainly about the whole process. The mixes should be grouped in groups of two and the selected outputs of the first mix shouldn't be the selected inputs of the second mix. This hides the relation between the input of the first mix and the output of the second mix, see Figure 3.2. The probability of success of changing k messages by an adversary controlling some minority group of mix servers is $p = \frac{1}{2^k}$, so even for a low k the probabilities are not high. The low probability of changing a significant amount of votes without being detected and the simplicity of the RPC makes it an interesting approach to be use in large scale elections.

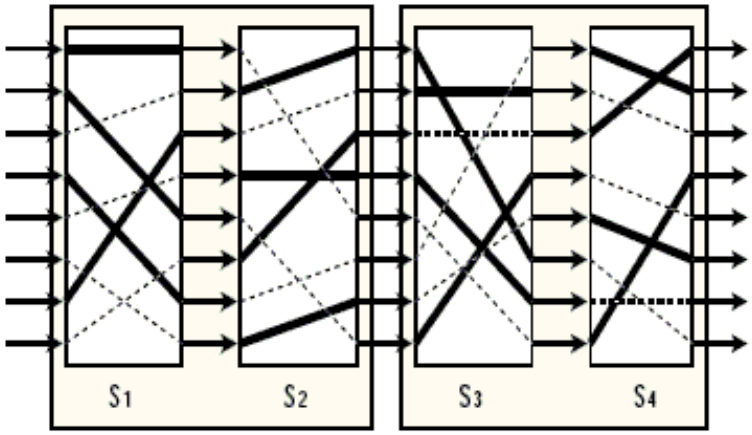

Figure 3.2: Randomized Partial Checking

Chapter 4

Related work

In this Chapter we start with an analysis and evaluation of electronic voting protocols.

Then we present another analysis and evaluation of implementations of electronic voting systems based on blind signatures.

4.1 Voting protocols

In this Section we start by presenting some simple voting protocols [Cra96]. Then we describe more sophisticated voting protocols that make use of some cryptographic techniques described in Chapter 2. Finally it is presented an overall evaluation of the protocols and some conclusions.

4.1.1 Simple protocol

We could imagine a simple voting protocol with only two authorities that seems to satisfy the required properties for voting systems. The two authorities are a *validator* and a *tallier*. The protocol is presented in Figure 4.1 and goes as follows: (1) a voter submits his ballot and identification (ID) to the *validator*; (2) the *validator* checks if the voter had already voted, if not

it detaches the voter ID from the ballot and sends the ballot to the *tallier*; (3) finally, after the election closing, the *tallier* performs the tally.

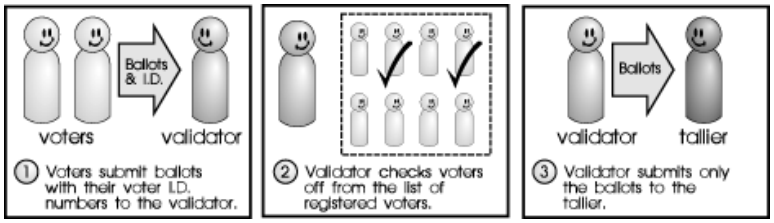

Figure 4.1: Simple voting protocol

This protocol has several major problems, namely: (1) even with the use of some cryptography to assure that only the *tallier* can see the ballots, a simple collusion between the two authorities result in a break of voters' privacy; and (2) nothing prevents the insertion/substitution of ballots by the *validator* and *tallier*.

4.1.2 One/Two agency protocols

To solve the above problems, in 1991, Nurmi, Salomaa and Santean proposed the Two Agency Protocol [NSS91]. In this protocol, described in Figure 4.2, we also have a *validator* and a *tallier* but now the voters interact with both authorities. The protocol goes as follows: (1) the first step is a distribution of ID TAGs to the voters by the *validator*; (2) then the *validator* sends a list of ID TAGs to the *tallier* without revealing the relation between the real voters ID and the ID TAGs; (3) each voter then submits, to the *tallier*, an encrypted message containing the ID TAG and the ballot; (4) finally the *tallier* publishes the list of received messages, and the voters send the decryption keys to the *tallier*. At this time the *tallier* can perform the tally.

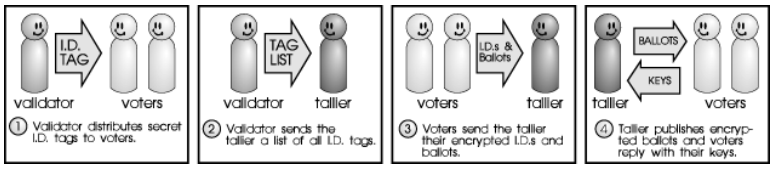

Figure 4.2: Two Agency Protocol

The main problem of this Two Agency Protocol is that the voters' privacy can be easily compromised by the collusion of the two authorities. The authors state that if the two agencies are going to work together, there might as well be just one agency.

The One Agency Protocol, proposed in 1991 by Salomaa [Sal91], is identical to the Two Agency Protocol, except in the tag distribution phase. To distribute the tags it's used the ANDOS (All-or Nothing Disclosure Of Secrets [BCR87]) protocol for secret selling of secrets. This solves the collusion problem, but now we must trust in one all mighty agency. Furthermore, the ANDOS protocol is very heavy computationally and doesn't scale well.

4.1.3 Blind signatures

The blind signature approach makes use of a simple technique that requires a small amount of computational power. Also it is independent of the ballots format, which is good if we want to use the voting system in non conventional elections, such as open-answer surveys.

There are several proposed voting protocols based on the blind signatures scheme [CC97, Cha88, DuR99, FOO92, Her97, HS98, Oka96, Oka97, OMA+99]. The reference protocol in this type of voting systems, known as the FOO protocol, was proposed in 1992 by Fujioka, Okamoto and Ohta

[FOO92]. We will now present the steps of the protocol involving the voters, an *Administrator* and a *Counter*.

1. **Preparation:** The voter fills in a ballot and encrypts it with a random secret key k. Then he constructs a message by blinding the encrypted ballot (cf. Section 3.3.1). Finally, he sends the message to the *Administrator* for signing.

2. **Administration:** The *Administrator* signs the message containing the voter's hidden ballot, and returns the signature to the voter.

3. **Voting:** The voter removes the blindness from the signature, and anonymously sends the ballot and the signature to the *Counter*. Note that the *Administrator* cannot link the signatures it provided with the signatures that the voter gets after applying the unblinding factor (cf. Section 3.3.1).

4. **Collecting:** The *Counter* publishes a list of received encrypted ballots. Since the encrypted ballots are submitted anonymously the voters could submit their vote several times (the repeated encryptions are considered the same vote).

5. **Opening:** Each voter sends his decryption key k anonymously.

6. **Counting:** The *Counter* counts the votes and announces the result.

In the FOO protocol the voter cannot vote and walk away. He must wait until the end of the election to submit his key, therefore bringing no convenience to the voter. There have been some implementations of protocols closely based on the FOO protocol, two of the most well-known are: the Sensus protocol [CC97], proposed by Cranor and Cytron, and the Evox protocol

[Her97] proposed by Herschberg, both in 1997. In both proposals the FOO protocol was changed to allow the voter to vote and walk away (see details in Section 4.2).

Later, in 1999, another improvement to the FOO protocol was proposed by Ohkubo *et. al.* [OMA+99]. Besides addressing the vote and walk away problem, they also propose the use of a (t, N) - threshold cryptosystem to prevent the leak of intermediate results. In their protocol the voter encrypts his signed ballot and sends it anonymously to a bulletin board. Then $n \geq t$ counters cooperate to produce the final tally. With this solution it is needed t colluding counters to leak intermediate results.

One of the unsolved problems of the previously presented protocols is **robustness**. In 1999, DuRette proposed the Evox-MA [DuR99], an improvement of Evox by the addition of more *Administrators*. The improvement addressed the problems of collusion resistance and availability. However, the use of a weak voter's authentication mechanism makes voter's impersonation by the *Administrators* easy, therefore leading to a much less robust system than the author claims (see details in Section 4.2.3).

None of the above referred protocols is receipt-free. This property was addressed by Okamoto in 1996 [Oka96] and 1997 [Oka97]. The first proposal had a problem that Okamoto solved in the second paper. To guarantee the receipt-free property in his proposals, Okamoto assumes the existence of some physical requirements, such as untappable channels[1] or private voting booths[2].

The previously mentioned protocols have in common the use the blind signature technique to sign the ballots. In 1998, Q. He and Z. Su proposed

[1] An untappable channel is a physical apparatus by which an entity A can send a message to another entity B, maintaining the message perfectly secret to all other entities.

[2] A voting booth is a physical apparatus by which a voter V can interactively communicate with a party, maintaining the communication perfectly secret to all other parties.

a different approach, where each voter should have a secret signed election key [HS98]. The protocol goes as follows:

1. **Registration:** In this phase the voter generates an asymmetric key pair and, using the blind signature technique, obtains a signature on the public key.

2. **Public key submission:** Each voter submits anonymously his signed public key to the *Counter*. The *Counter* verifies the authenticity of the key verifying the signature on it.

3. **Voting and tallying:** Each voter fills a ballot, signs it with his private key and sends it anonymously to the *Counter*. The *Counter* verifies the signature using the signed public key, and if all is correct he proceeds with the tally.

This protocol suffers from the problem of allowing the leak of intermediate results. It has also a problem with the re-usability of the data collected in the registration and public key submission phases. Because there is no connection between the voters and the signed public keys in an election ***e1,*** it's not possible to use the data collected in ***e1*** in another election ***e2*** having a different electorate (could be just a voter less), because we cannot eliminate the keys of specific voters. This last problem is a major setback to the use of the system in large scale elections, which almost always have a different electorate. Enforcing a registration process in every election is neither practical nor convenient for voters.

4.1.4 Mix-Nets

In 1981, David Chaum proposed the first election system based on mix-nets [Cha81]. The protocol goes as follows:

1. **Preparation:** In this phase it's made a registration of the voters where it's generated a digital pseudonym[3] for each voter. In this phase are also published the public keys of all mix servers on a BB.

2. **Voting:** Each voter fills his ballot and signs it with his digital pseudonym. Then successively encrypts the ballot with the keys of the mix servers (cf. encryption mix-nets in Section 3.6). Finally, the voter sends his vote to the BB.

3. **Decryption:** This phase works as the decryption on encryption mix-nets, described in Section 3.6. At the end all signed ballots are published on the BB.

4. **Tallying:** The signatures on the ballots are verified and all valid ballots counted.

To protect the voters privacy it is used digital pseudonyms, however this approach makes impossible to reuse the pseudonyms in another election if the electorate change, namely if any removal of voters is needed. This is the same problem of the Q. He and Z. Su blind signature proposal, therefore also a major setback to the use of the system in large scale elections, which almost always have a different electorate.

The Chaum's protocol has also the problem of ciphertext length expansion, inherited from the used mix-net construction. In 1993, Park, Itoh and Kurosawa [PIK93] proposed a protocol (PIK protocol) based on re-encryption mix-nets that does not suffer from such a problem. The PIK protocol is also a all/nothing election scheme, i.e., or every vote counts or no vote counts. This property is achieved by dividing the ballots in two pieces.

[3] A digital pseudonym is a public key used to verify signatures made by the anonymous holder of the corresponding private key.

In the decryption phase it is first decrypted one piece, selected at random, and then decrypted the second piece of the ballots. If some disruption is detected in some decryption step the protocol stops at once and no tally is published.

In 1995, Sako and Killian proposed the first receipt free mix-net based voting system [SK95]. In their protocol the voter, instead of filling up his ballot, chooses one from a list of already filled ballots. Here are steps of the protocol:

1. **Registration:** In the registration step all voters are registered with their public keys.

2. **Choosing a vote:** In this step the voters interact with several authorities to choose a previously filled ballot, see Figure 4.3. The first authority shows all filled ballots to the voter using an untappable channel. Then encrypts the ballots and secretly permutes them. The proof of correctness of all the operation in made using a commitment. The commitment was previously established with the voter using the voter's public key. Finally the first authority passes the encrypted ballots to the next authority. The process is the same for all other authorities until the last authority is reached. At the end of this process only the voter knows the relation between the original ballots and the final encrypted ballots.

3. **Submitting the ballot:** The voter submits the desired encrypted ballot to the first mix of the mix-net.

4. **Tallying:** After the last mix reveals the permuted ballots, everyone can compute the final tally.

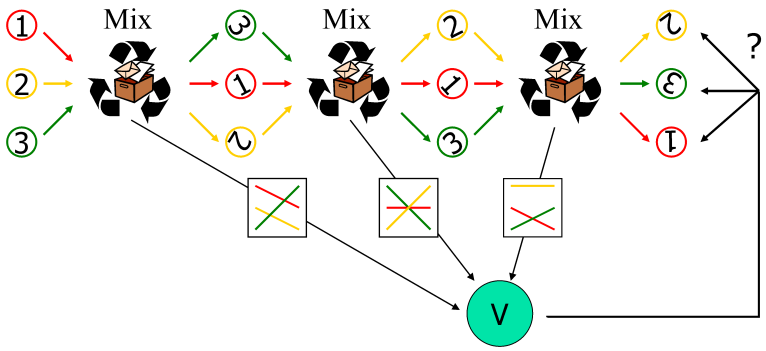

Figure 4.3: Anonymous ballot choosing using a mix-net.

The previously mentioned protocols stop if only one of the mix servers stops. In 1997, Ogata *et. al.* [OKST97] presented a solution that solves the problem, tolerating less than half faulty servers. In their protocol each mix server shares its private key with all other mix servers in a (t,N)-threshold way, where t is half of the total number of servers. With the sharing of mix server's private keys it is possible to recover the private key of any mix server that had failed during the protocol, as long as t mix servers remain working correctly. Therefore the protocol can be repeated without the participation of the faulty mix servers.

The main problems with mix-net based voting systems are the complexity of the correct permutation proof, and the tolerance to mix servers' faults. There is work made in the last years concerning the efficiency and robustness of mix-nets [Abe98, Abe99, AH01, BG02, FS01, Jak98, Jak99, JJ01, MK00, Nef01, JJR02]. This work could be used to improve the previously mentioned protocols.

4.1.5 Homomorphic

The homomorphic based voting systems, as the name suggests, make use of the homomorphic property of some cryptographic systems. The main characteristic that distinguishes this type of system from the others is that at the end individual votes are not decrypted; is the final result of the tally that is decrypted. In other words, individual votes are not observable, only the final tally of all votes can be computed and published.

The initial work in this field was done by Benaloh *et al.* [BF85, BY86, Coh87]. The model assumes l voters $V_1, ..., V_l$, a BB and n tallying authorities (talliers) $A_1, ..., A_n$. It's used a (t, N)-*threshold cryptosystem* so that the tally could only be performed with the cooperation of t talliers. The voting process in homomorphic election systems normally involves three phases:

1. **Preparation:** In this phase is made the registration of the voters. It is usually assumed the use of a public encryption to authenticate the voters. The talliers generate the election key pair, in a (t, N)-*threshold* way, and publish the public part in the BB.

2. **Voting:** In the voting phase the voter first fills his ballot, encrypts it with the election key, and signs it with his private key. Then the voter contacts the BB, authenticates himself and sends the encrypted signed ballot. The BB publishes the encrypted signed ballot. Because the ballots will not be decrypted individually, the voter must prove in a zero knowledge way that what he had send to the BB is an encryption of a valid vote. The proof is also maintained in the BB.

3. **Tallying:** To produce the outcome of the election at least t talliers must work together. First, using the homomorphic properties, they compute the encryption of the tally. Finally, without leaking anything

about the private key of the election, the talliers jointly decrypt the tally and prove of the correction of it in a zero knowledge way.

Homomorphic ciphers and zero knowledge proofs are rather complex, therefore the work done in the past years in these kind of systems tackled mainly the complexity of the systems [BFP+01, CFSY96, CGS97, DJ01, SK94]. Most of the voting schemes are based in a variant of the ElGamal encryption scheme proposed by Cramer, Gennaro and Schoenmakers in [CGS97]. This scheme is efficient but only for "yes/no" votes. It is also restricted to small or medium scale elections because of the computational complexity of the homomorphic cipher. To support a multi-candidate election without using a "yes/no" vote for each candidate, Baundron *et al.* in [BFP+01] employed the Paillier cryptosystem in the construction of a practical multi-candidate voting system.

The receipt-free property has also been a target for researchers [BT94, HS00, LK02]. In 2000, Hirt and Sako in [HS00] proved that the scheme proposed in 1994 by Benaloh and Tuistra [BT94] was not receipt-free. Hirt and Sako's proposal complements the work of Cramer *et al.* [CGS97] with the work of Sako and Killian [SK95] to construct a receipt-free voting system. Their system uses untappable channels between the authorities and the voters. The voting protocol is similar to the one described above except in the voting phase, where for choosing a ballot it is used a similar construction to the one proposed by Sako and Killian, see Figure 4.3.

In 2002, Lee and Kim [LK02] proposed a receipt-free voting system that doesn't require the use of an untappable channel. They propose to use a tamper-resistant device, such a smart card or Java card, to play the role of the untappable channel.

	Homomorphic based	Blind Signatures based	Mix-Nets based
Mathematical structure	much	little	medium
Voter interactions	1	>1	≥1
Tallying	Decryption of the totals	Decryption of individual votes	Decryption of individual votes
Costs:			
Voting	high	small	medium
Tallying	small	very small	medium
Verification	small	local only	Medium
Assumptions and restrictions	Restrict ballot format	Anonymous channels or private voting booth	Some ballot restrictions
Scalability	medium	good	medium

Table 4.1: Electronic voting protocols evaluation

4.1.6 Overall evaluation and conclusions

We have presented an overview of the electronic voting protocols and the technologies used by them. The simple and one/two agency protocols offer few guarantees on electronic voting properties. This leaves only three promising types of electronic voting systems: the ones based on blind signatures; the ones based on mix-nets; and the ones based on homomorphic ciphers. We will now present an overall evaluation of these systems, resumed in Table 4.1.

The systems using blind signatures have the advantages of having lower costs (computational and network traffic), being simpler and entirely ballot independent. On the other hand, they require more interactions with voting authorities (servers supporting elections) than the other systems: at least one for each required signature and one to cast the ballot. Another problem is the assumption of the existence of an anonymous channel for hiding the address or any other identification of voter's machines.

Proving the correctness of mix-nets is the main problem of the mix-nets

approach. The zero knowledge proofs are complex and have high costs, both computationally and also in terms of space. Some work as been done to produce more robust and efficient mix-nets. We believe that most of this work can be brought back to voting systems. Another problem of mix-net based voting system is the need of a new voters' registration every time the electorate change. The positive aspects of mix-net based voting systems are the reduced number of interactions between the voter and the election servers and good ballot independence.

The homomorphic based voting systems are the most complex systems in mathematical terms, because they require homomorphic ciphers. This also brings the disadvantage of a high computational cost. Also due to the homomorphic cipher requirements, there are major ballot restrictions. To keep the computational cost as low as possible, only yes/no answers are allowed. To support 1-of-n answers it's commonly used a yes/no answer for each possible answer. The positive aspects are the need of only one voter interaction and the confidentiality of individual votes.

The four core properties - accuracy, democracy, privacy and verifiability - with the exception of the second part of the privacy property, receipt-freeness, are commonly reached by all proposals. The proposed schemes that are receipt-free make use of physical devices like untappable channels, private voting booths, smart cards, java cards or some other devices. But if we are talking about using totally remote Internet voting systems, it is impossible to entirely fulfill the privacy property: it will be always possible to a voter to prove that he voted in a particular way, e.g. the voter can vote in the presence of others, or it can allow someone to peek on his computer using some program like the BackOrifice [BO2].

As shown so far in this Chapter, there is much work developed around

electronic voting. There are mainly three promising voting approaches to tomorrow's voting systems, each one with its pros and cons. For REVS we have chosen a blind signature approach because of its intrinsic advantages: simplicity, low costs (it is computational and network efficient) and ballot independence.

4.2 Voting systems

Despite the fact that there are many protocols proposed for electronic voting the implementations are few. Since we choose the blind signatures approach for REVS, we present in this Section an evaluation of four implemented voting systems based on blind signatures, namely Sensus, Evox, Evox-MA and the Vienna's system.

In our analysis we will first present the four systems, then we will evaluate the systems concerning the properties defined in Section 2.2. In particular we will evaluate:

- The possibility of modification and elimination of valid votes (Accuracy).
- The possibility of an invalid vote to be part of the tally (Accuracy).
- The possibility of obtaining a valid vote from anyone who is not an eligible voter, and also the possibility of a double voting from an eligible voter (Democracy).
- The possibility of partial disclosure of results (Democracy).
- The possibility of connection between the vote and the voter by any authority running the election, and also by the voter (Privacy).

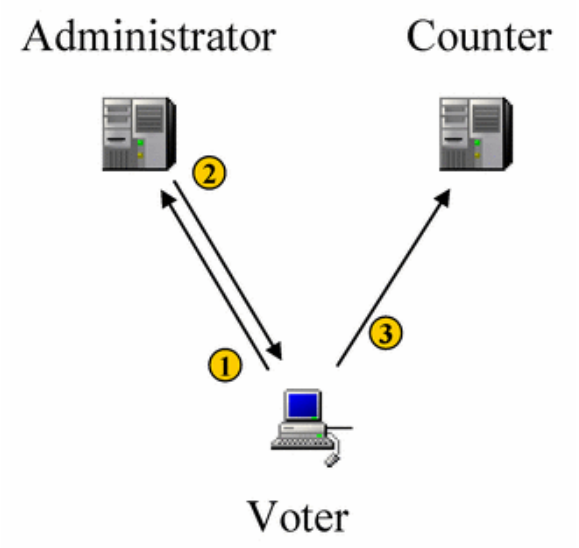

Figure 4.4: Simple blind signature voting system

- The Verifiability of the tally.
- How the system supports the voting process in the presence of servers failures, or servers network connection problems (Availability).
- The possibility to resume the voting protocol after unexpected failures that prevent the voter from ending the voting process, such as: network failures, computer crash due to hardware or software problems, etc. (Resume-ability).
- The protection of the core properties against a collusion of any entities running the election (Collusion Resistance).

Simple blind signature voting system

To introduce the terminology used, and also to remember how a blind signature voting system works, we will start by explaining a simple blind signature voting system (see Figure 4.4).

- First the voter fills the ballot.
- Then the voter blinds the ballot using a blind factor and sends the blinded ballot to the *Administrator* for signing (1).
- The *Administrator* verifies if the Voter did not vote yet; if not signs the blind ballot and returns it to the voter (2).
- The voter removes the blinding encryption layer from the ballot and verifies the signature of the *Administrator*. Only a valid signature can be used to produce a valid vote.
- At last the voter anonymously submits his ballot signed by the *Administrator*, to the *Counter* (3).
- After the end of the election the *Counter* verifies the signatures on the votes, counts the valid votes and publishes the final tally.

Now that we established this terminology, it is time to present the systems under analysis.

4.2.1 Sensus

Sensus was proposed by Cranor and Cytron in 1997 [CC97], and was one of the firsts implementations of the FOO protocol. As mentioned in Section 4.1.3, the authors of Sensus modified the original voting protocol of FOO to allow a vote and walk away process. In Sensus it was kept the authentication method proposed in FOO, i.e. using public key authentication. The steps of Sensus protocol are the following (see Figure 4.5):

- First the voter fills the ballot, encrypts it using a secret key, and blinds it.

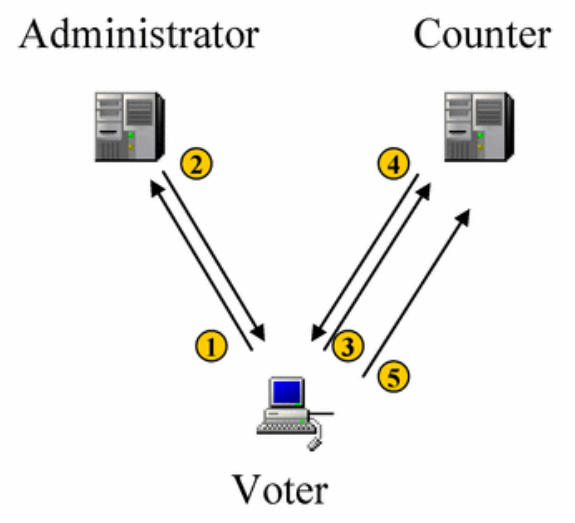

Figure 4.5: Sensus voting system

- The voter then signs the blind ballot and sends it to the *Administrator* (1).
- The *Administrator* verifies that the signature belongs to a registered voter who had not yet voted. If the *Administrator* had not signed before for that voter it signs the blind ballot and returns it to the voter (2).
- The voter removes the blinding encryption layer, revealing an encrypted ballot signed by the *Administrator*. The voter then sends the signed encrypted ballot to the *Counter* (3).
- The *Counter* checks the signature on the encrypted ballot and, if valid and if the encrypted ballot is unique (as in FOO protocol), the *Counter* places it on the list of valid ballots to be published at the end of the election. The *Counter* then signs the encrypted ballot and returns it to the voter as a receipt (4).

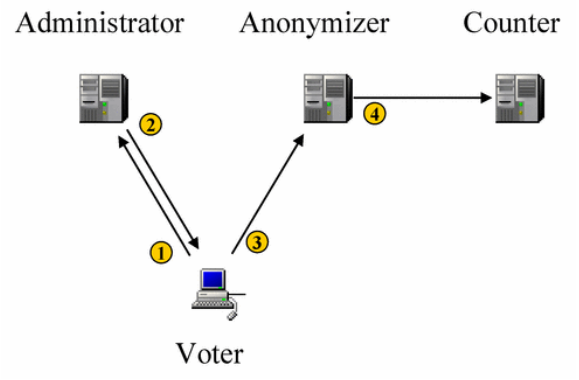

Figure 4.6: Evox voting system

- Upon receiving the receipt, the voter sends to the *Counter* the ballot decryption key (5). Finally, the *Counter* uses the key to decrypt the ballot and adds the vote to the tally.

In Sensus the step for submitting the decryption key of the FOO protocol was anticipated. Now the voter submits his decryption key before the end of the election. With this modification the voter is able to vote and walk away, but now the *Counter* is also able to leak intermediate results as the voters submit their decryption keys. The *Counter* can also can link the vote to the machine used to submit it.

4.2.2 Evox

The Evox protocol was proposed in 1997 at MIT by Herschberg as his Master Thesis [Her97]. The Evox system is another implementation based on the FOO protocol. Like Sensus the voting protocol was adapted to allow the voters to vote and walk away. The Evox voting protocol is the following (see Figure 4.6):

- First the voter obtains the ballot from the *Administrator*. Then he

must fill the ballot and commit to it using a random bit string.

- Then he blinds the committed ballot and sends it to the *Administrator* for signing (1).

- The *Administrator* verifies if had not already signed for the voter, and if not it signs the blinded ballot, updates the voter record to an already voted state and returns the signed blinded ballot to the voter (2).

- After receiving the signed blinded ballot the voter removes the blinding layer and obtains a signed ballot. To complete the voting process, the voter encrypts the ballot, the bit commitment and the signature with the public key of the *Counter*; then he anonymously sends the encryption to the *Counter* through the *Anonymizer* (3). The voter can submit the vote as many times as he wants.

- When the election ends the *Anonymizer* forwards the encrypted votes, in a random order, to the *Counter* (4).

- Finally the *Counter* decrypts the votes, removes the repeated votes by verifying the random commitment bit string and produces the election tally.

Both Sensus and Evox modify the voting protocol of FOO to allow a voter to vote and walk away. While Sensus only anticipates the decryption key submission step of the FOO protocol, in Evox there was made a more substantial change in the voting protocol. In Evox each voter encrypts his vote with the *Counter's* public key, therefore the decryption key submission step is not necessary. Since the encryption of the ballot was also used for detecting repeated ballots, there was the need to create another way to detect

repeated ballots. In Evox, to detect repeated votes, it is used a bit commitment performed by the voter. All votes with the same bit commitment are considered the same, therefore only one will be in the final tally.

Another difference from Sensus and FOO is the use of the username/password authentication. Despite the weakness of the username/password authentication compared to the public key authentication, it still as a major advantage: common knowledge, today every one uses username/password authentication for accessing email accounts, e-banking, etc.

4.2.3 Evox - Managed Administrators

The Evox - Managed Administrators (Evox-MA) was proposed by DuRette, in 1999, as his bachelor's thesis [DuR99]. DuRette's work tackles a common problem in voting protocols based in blind signatures: preventing the democracy corruption by the *Administrator*. Since an *Administrator's* signature is the base requirement to make a ballot valid, nothing prevents the *Administrator* from creating and submitting illicit valid ballots. The *Administrator* can also prevent a voter from voting, refusing to sign his ballot, or allow several votes from the same voter, signing several times for the voter.

The idea explored by DuRette is to ensure democracy by sharing the power of the *Administrator* among several servers. In DuRette proposal there are n *Administrators*, and t signatures of them are required to make a ballot valid. There was also introduced the *Manager* server that will sign the list of t signatures of the *Administrators* to allow $t \leq n/2$ (see the discussion ahead).

The protocol is similar to the one of Evox, and goes as follows (see Figure 4.7):

- First the voter fills a ballot and commits to it using a random bit

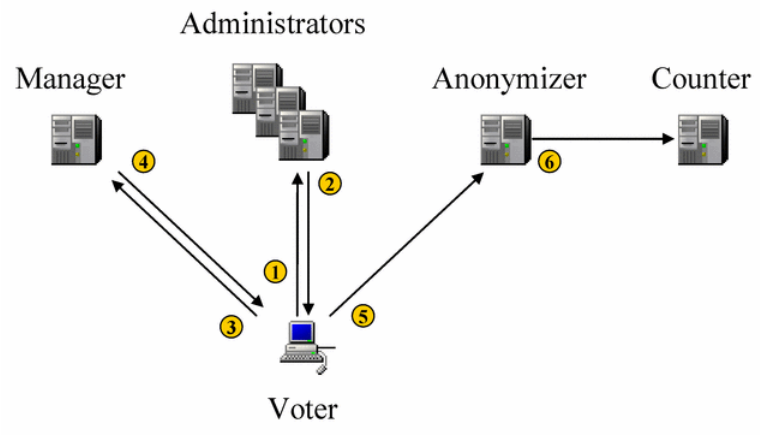

Figure 4.7: Evox-MA voting system

string, as in Evox. Then he blinds the committed ballot and sends it to $t \leq n$ *Administrators* for signing (1). In the Evox-MA case the ballot is obtained from the *Manager*.

- Each *Administrator* verifies independently if had not already signed for the voter, and if not they sign the blinded ballot, update the voter record to an already voted state and return the signed blinded ballot to the voter (2).

- After receiving all blinded signatures the voter removes the blinding layer and obtains a list of t signatures on the ballot. At this point the voter blinds the list of signatures and sends it to the *Manager* (3).

- If the *Manager* had not already signed for the voter, he signs the blinded signatures list and returns it to the voter (4).

- The voter receives the blinded signed signature list and unblinds it. Finally, to complete the voting process the voter encrypts the ballot, the bit commitment, the signatures and the signature on the list of

signatures with the public key of the *Counter* and he anonymously sends the encryption to the *Counter* through the *Anonymizer* (5).

- When the election ends the *Anonymizer* forwards the encrypted votes to the *Counter* in a random order (6).

- The *Counter*, after receiving the encrypted votes, decrypts them, removes the repeated votes and process the election tally.

In Evox-MA the democracy property is guaranteed by the *Administrators* and the *Manager* if they sign only once per voter. If $t \leq n/2$ the voter can get distinct lists of t signatures, therefore in this case is the *Manager* that prevents a voter from obtaining more than one valid ballot. If $t > n/2$ the voter can only obtain of list of t signatures, therefore the *Manager* is not needed to guarantee democracy.

Apparently, in Evox-MA it is needed the collusion of t *Administrators* and the *Manager* to introduce valid votes. However, Evox-MA does not offer the apparent collusion-resistance because it is used only one password per voter for all *Administrators* and also for the *Manager*. None of these entities knows the password in advance, because a UNIX-like validation is used, i.e. the entity only have the digest of the password and not the real password. However, a small set of *Administrators*, in collusion with the *Manager*, can generate illicit valid votes using the voter's password once they get it. The fraud may work like this: x colluded *Administrators* use the voter's password to get signatures from all the *Administrators* not yet contacted by the voter. Then they send to the *Manager* a signed vote that he could accept and send to the *Counter*. With n *Administrators* and $n/2 + \Delta$required signatures, x is equal to 2Δ. If, for improving performance, Δ is a low value (1 or 2), the possibility of attack is not negligible. If t is less than $n/2$, the *Manager* itself

can introduce votes without the participation of any other entity.

In Evox-MA there is some resistance to failures and collusion. The voter must get t signatures from the *Administrators* and one from the *Manager*; therefore the voter can lose up to $n - t$ signatures from the *Administrators*, or tolerate the failure of them, without been prevented to vote. However if the voter loses the *Manager* signature before submitting the vote, then he will be prevented to vote because the *Manager* only signs once.

We can say that the robustness of Evox-MA is higher than the one of Evox, but due to a weak authentication protocol it is not as good as it could be.

4.2.4 Vienna's e-voting system

Kofler, Krimmer and Prosser presented, in 2003, a protocol that is the basis of an e-voting system developed at the Vienna University for Business Administration and Economics (Austria) [KKP03]. Their proposal is based on the blind signature technique and strictly separates two phases in the voting protocol, the registration phase and the voting phase.

The registration phase in open for an arbitrary period of time before the election day. Besides the usual voters' registration in this system each voter will also validate two tokens (see Figure 4.8):

- First the voter generates a random token t and blinds it. Then the voter signs the blinded token and sends it to the *Registration* server (1).

- The *Registration* server verifies if the signature belongs to a valid voter, if so the *Registration* server signs the blinded token t and returns it to the voter (2).

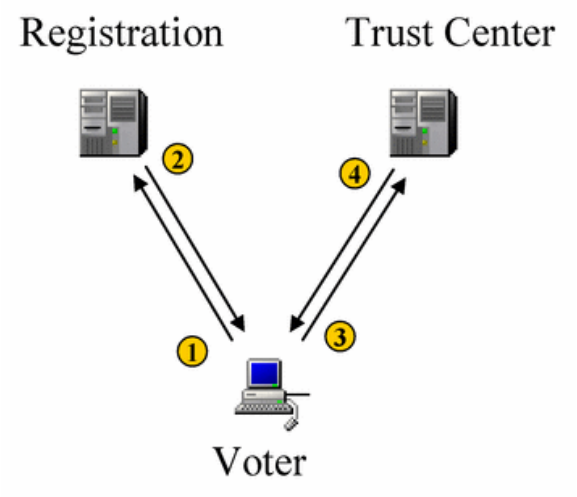

Figure 4.8: Voter registration

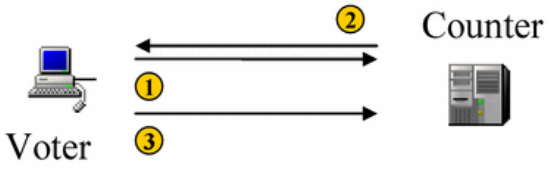

Figure 4.9: Voting phase

- A similar process is repeated with the *Trust Center*: the voter issues a second token τ, blinds it and obtains the signature on it from the *Trust Center* (3 e 4).

If any token is lost the voter can re-apply for another token, in this case the servers will respond with the original signed token to avoid issuing multiple tokens. At the end of the registration phase the voter holds two signed authentication tokens, both needed to cast a vote on election day. Both the *Registration* server as the *Trust Center* can be seen as two *Administrators*.

The voting phase is processed in the election day (see Figure 4.9):

- The voter sends the signed tokens to the *Counter* server to obtain his

ballot (1).

- The *Counter* checks the signatures on the tokens and if they verify sends an empty ballot to the voter (2).
- Then the voter fills the ballot and sends it, with the signed tokens to the *Counter* (3).
- At the end of the election day the *Counter* publish the filled ballot and the corresponding signed tokens.

In the voting phase the authentication of the voters is only based on the previously signed tokens. Therefore, the authors claim that is harder to link the ballot to the voter by the network address of the voter's PC. But this is only true is the voter uses different hosts for the registration and for the voting.

In this voting system there are no strong link between the ballot and the tokens. Therefore it is possible to the *Counter* to change any ballot without being detected.

As the FOO protocol, this system also requires two connections apart in time from the voter; therefore it is not convenient for voters.

4.2.5 Evaluation and conclusions

Now that we have presented the four systems, we will proceed with the evaluation of them.

Accuracy

In all systems the voter can detect if his vote was eliminated from the tally. If the fraud is detected he can anonymously resent his vote to correct the tally.

		Sensus	EVOX	EVOX-MA	Vienna's voting system
Accuracy	Elimination of votes	Detect and correct	Detect and correct	Detect and correct	Detect and correct
	Introduction of invalid votes	Detect and correct	Detect and correct	Detect and correct	Detect and correct
	Change vote	Invalidate vote	Invalidate vote	Invalidate vote	Possible with no correction

Table 4.2: Accuracy analysis resume

The introduction of invalid votes to the tally can be detected in all systems by verifying the signature on the ballots, or on the tokens in Vienna's system case. Therefore, if a signature does not match a vote, it is removed from the tally.

In Sensus, Evox and Evox-MA it is impossible to change a vote, any change in the vote will ruin the signatures, therefore invalidating the vote. In Vienna's system it is possible to alter a vote, because a change in the vote (ballot) will not ruin the signatures on the tokens, therefore we would still have a valid vote. Also this fraud cannot be corrected by the voter; he cannot prove that the published vote is not his, because there is no connection between the vote and the signed tokens.

The Sensus, Evox and Evox-MA systems respect all aspects of the accuracy property. The Vienna's system has a drawback because it allows changes in the vote without invalidating it.

Table 4.2 presents a resume of the accuracy analysis.

Democracy

In all systems all voters are able to vote exactly once if the *Administrator(s)* or equivalent servers work properly. In the case of misbehaviour from the *Administrator(s)* to prevent a voter from voting (e.g. signing garbage

instead of the voter request, or simple refusing to sign), there is a main difference between the systems using public key authentication, Sensus and Vienna's, and the ones using username/password authentication, Evox and Evox-MA. When it is used public key authentication, the voter protest is verified checking the existence of a voter's signed request in the *Administrator's* logs; if there is no such record it the *Administrator's* fraud is proved, otherwise is the voter who is trying to mess with the system. In the case of username/password authentication the voter protest is handled as previously, however if there is a record it is impossible, for the voter and also for the *Administrator*, to prove that the record is an original voter's request. In Evox-MA case this kind of misbehaviour is tolerated in up to $n - t$ *Administrators*, because for a valid vote there are needed t signatures of different *Administrators*; however the problem subsists for the required signature of the *Manager*.

In blind signature voting systems the *Administrators* are the entities with the power of making a vote valid, therefore they can produce valid votes and send them anonymously to the *Counter*. The detection of this fraud can be made verifying if the number of votes in the tally is higher than the number of signatures in the *Administrators'* lists. Despite the easy detection of this fraud it cannot be corrected because the votes are sent anonymously. In Sensus and Evox systems this fraud can be performed by the *Administrator* alone. The Vienna's system uses two "*Administrators*", the *Registration* and *Trust Center*, therefore it would require the cooperation of these two servers to perform the fraud. In Evox-Ma system we encounter a general approach for any number of *Administrators* but, as explained in Section 4.2.3, due to a week authentication mechanism the number of required *Administrators* to perform the fraud is much less than it could be.

		Sensus	EVOX	EVOX-MA	Vienna's voting system
Authentication		PKI	Username/ Password	Username/ Password	PKI
Democracy	Prevent voter from voting	Detect and correct	Administrator	Manager or *(n-t* +1) Administrators	Detect and correct
	Introduction of valid votes	Administrator	Administrator	Manager and $t \leq$ Administrators	Registration and Trust Center servers
	Voting for absentee voters	No	Administrator	Manager and t Administrators	No
	Leak of partial results	Counter	Counter and Anonymizer	Counter and Anonymizer	Counter

Table 4.3: Democracy analysis resume

There is also another fraud that can pass undetected, the voting for absentee voters. This kind of fraud can only be performed if the system uses a username/password authentication, because in this case the *Administrators* can introduce a valid entry on their logs. This fraud can be performed as follows: near the end of the election period the *Administrators* produce valid votes for the absentee voters, update the signature logs, and anonymously submit the votes to the *Counter*. This kind of fraud is minimized in the Evox-MA system, because it requires t signatures, therefore it would be necessary t colluded Administrators, plus the Manager, to perform the fraud.

The last item to analyse is the leak of partial results. In Sensus and Vienna systems the *Counter* can leak partial results as the voters submit their votes. In the case of Evox and Evox-MA systems, the votes are encrypted with the *Counter's* public key and are kept by the *Anonymizer* until the end of the election period. Therefore, the only way to leak intermediate results is with a collusion between the *Counter* and the *Anonymizer*.

Table 4.3 presents a resume of the democracy analysis.

		Sensus	EVOX	EVOX-MA	Vienna's voting system
Privacy	Anonymous channel	Need	Uses Anonymizer	Uses Anonymizer	Need if the voter uses same PC
	Collusion to break anonymous channel	Administrator and Counter	Administrator, Anonymizer and Counter	Administrator or Manager, Anonymizer and Counter	Registration or Trust Center and Counter
	Receipt-free	No	No	No	?

Table 4.4: Privacy analysis resume

Privacy

In blind signature voting systems the privacy of the voters relies on the existence of an anonymous channel between the voters and the *Counter*, to prevent an attack based on the network address of the voter's host. In Evox and Evox-MA it is proposed the use of anonymizing servers as anonymous channel providers. The Sensus authors do not provide details about any particular anonymous channel, they only assume the existence of one. The Vienna's system tries to protect voters' privacy using a two phase protocol, in which the voters can use two different hosts, one to register and another to vote.

None of these systems is receipt-free, therefore the voter as a receipt at the end that can be used to prove the voter's choice. In the case of Sensus this receipt is the signature of the *Counter* on the encrypted vote; in Evox and Evox-MA the receipt is the bit commitment, and in the Vienna's system the receipt is the set of signed tokens. Since in Vienna's system the tokens are not directly linked to the vote, as explained in the accuracy analysis, the tokens can only be seen as a partial receipt, they do not prove with 100% of certainly that a particular voter voted in a particular way.

A resume table of the privacy analysis is presented in Table 4.4.

		Sensus	EVOX	EVOX-MA	Vienna's voting system
Verifiability	Own vote	Yes	Yes	Yes	Yes
	All votes	Signatures verification	Signatures verification	Signatures verification	Signatures verification?

Table 4.5: Verifiability analysis resume

Verifiability

In Sensus, Evox and Evox-MA anyone can verify the validity of the votes just by checking the validity of the *Administrators'* signatures. Voters can also detect and correct any mistakes with their votes without sacrificing their privacy, by resending anonymously their vote.

In the Vienna's system it is also possible to anyone the verification of the signatures on the two tokens but, as explained in the accuracy analysis, votes can be modified. Therefore, there is no effective mean to verify the correctness of the system.

Table 4.5 presents a resume of the verifiability analysis.

Robustness

- **Availability** - In all systems there are single points of failure; therefore the availability of all systems can be easily compromised. The only system who can handle some servers' failures is the Evox-MA: it can tolerate simultaneous failure of up to $n - t$ *Administrators*. However, there still exist singular points of failure, namely the *Manager* and the *Anonymizer*.

- **Resume-ability** - In Sensus and Evox if the voter looses his signed vote before submission there is no possibility of recovery, because the *Administrator* only signs once for each voter and keeps just a sign/not

sign record for each voter. In Evox-MA the voter can loose up to $n-t$ signatures of the *Administrators* because only t are required, but cannot loose the signature of the *Manager*. Only the Vienna's system has resume-ability: the *Registration* and the *Trust Center* servers keep the blinded signed tokens, therefore if a voter looses his tokens he can reapply for the tokens, the servers will return the previous signed tokens.

- **Collusion-resistance -** Here we present a resume on how systems stand for accuracy, democracy and privacy in the presence of colluded servers. If only one server alone is able to wreck a property we say that the system as no collusion-resistance concerning that property.

  Regarding the accuracy property, only the Vienna's system as problems: it allows the *Counter* to change votes. In what concerns democracy all systems have at least one aspect that can be ruined by only one server, therefore none of the systems can be considered collusion-resistant; however in an overall analysis of the democracy property we can consider the Evox-MA and the Vienna's systems more resistants to collusion than Sensus and Evox (cf. the democracy analysis).

  As said before in the privacy analysis, all systems require the use of an anonymous channel between the voters and the *Counter* to really protect the privacy of the voters. The Evox and Evox-MA proposed the use of an extra server, the *Anonymizer*, to provide the anonymous channel between the voter and the *Counter*; Sensus and Vienna's system do not propose the use of any special anonymous channel. We present the number of necessary colluding servers to break the privacy of the voters, using network address based attacks, in the resume Table 4.6.

		Sensus	EVOX	EVOX-MA	Vienna's voting system
Robustness	Availability	Single points of failure (all servers)	Single points of failure (all servers)	Single points of failure (except Administrators)	Single points of failure (all servers)
	Resume-ability	No	No	Only for *n-t* Administrators	Yes
	Collusion-resistance (accuracy)	Yes	Yes	Yes	No
	Collusion-resistance (democracy)	No	No	No	No
	Collusion-resistance (privacy)	2 servers	3 servers	3 servers	2 servers

Table 4.6: Robustness analysis resume

Concerning the robustness property, there is no satisfactory system. Every system has several single points of failure. For a system that must be available in a very restricted time frame, the existence of single points of failure is, at least, a very concerning problem. Only Evox-MA and Vienna's voting system have some tolerance to server and communication failures. Furthermore, an election can be easily disturbed in all systems by the misbehaviour/collusion of only one entity involved in the election; only the Evox-MA system tries to give a better protection against the collusion of entities but due to a weak voter's authentication protocol, the real protection in less than it could be.

Conclusions

In table 4.7 is presented a resume of the analysis presented before.

The first conclusion that we take from this analysis is that, in general, the solutions encountered satisfy the accuracy and verifiability properties. Regarding the privacy of the voters all systems are also similar: (1) they protect the privacy of voters if the servers are honest; otherwise, they require the use of an anonymous channel between the voters and the *Counter*; and

		Sensus	EVOX	EVOX-MA	Vienna's voting system
Authentication		PKI	Username/ Password	Username/ Password	PKI
Accuracy	Elimination of votes	Detect and correct	Detect and correct	Detect and correct	Detect and correct
	Introduction of invalid votes	Detect and correct	Detect and correct	Detect and correct	Detect and correct
	Change vote	Invalidate vote	Invalidate vote	Invalidate vote	Possible with no correction
Democracy	Prevent voter from voting	Detect and correct	Administrator	Manager or *(n-t* +1) Administrators	Detect and correct
	Introduction of valid votes	Administrator	Administrator	Manager and $t \leq$ Administrators	Registration and Trust Center servers
	Voting for absentee voters	No	Administrator	Manager and *t* Administrators	No
	Leak of partial results	Counter	Counter and Anonymizer	Counter and Anonymizer	Counter
Privacy	Anonymous channel	Need	Uses Anonymizer	Uses Anonymizer	Need if the voter uses same PC
	Collusion to break anonymous channel	Administrator and Counter	Administrator, Anonymizer and Counter	Administrator or Manager, Anonymizer and Counter	Registration or Trust Center and Counter
	Receipt-free	No	No	No	?
Verifiability	Own vote	Yes	Yes	Yes	Yes
	All votes	Signatures verification	Signatures verification	Signatures verification	Signatures verification?
Robustness	Availability	Single points of failure (all servers)	Single points of failure (all servers)	Single points of failure (except Administrators)	Single points of failure (all servers)
	Resume-ability	No	No	Only for *n-t* Administrators	Yes
	Collusion-resistance (accuracy)	Yes	Yes	Yes	No
	Collusion-resistance (democracy)	No	No	No	No
	Collusion-resistance (privacy)	2 servers	3 servers	3 servers	2 servers

Table 4.7: Overall systems analysis

(2) none is receipt-free.

We can also conclude that the use of public key authentication vs username/password authentication has major impact on the democracy property, namely in what concerns to the prevention of a voter to vote, and the voting for absentee voters. However, and despite the weakness of the username/password authentication when compared to the public key authentication, it still as the common knowledge advantage. The other aspects of the democracy property, namely the introduction of valid votes and the leak of partial results, need better protection against collusions of servers running the election.

None of the systems here analysed can be considered robust: (1) all of them have single points of failure; (2) in all systems a single server can ruin the democracy property; and (3) half of them do not have resume-ability.

In the real world it is usual to run multiple elections simultaneously. In Evox and Evox-MA systems it is possible to run multiple elections simultaneously, however this is a weak support; it is only possible to have simultaneous elections if each one has a different electorate. The multiple elections' support in Evox and Evox-MA is based on the ballot distribution: if a voter A is registered for the election E1 it will get the ballot EB1; if the voter B is registered in election E2 it will receive the ballot EB2. This process does not prevent the voters from exchanging votes, therefore from voting in elections that they should not vote. In the Vienna's system the ballot is obtained using the blind signed tokens, therefore with no connection to the real voter. Because of this characteristic there is no possible support for multiple elections. In Sensus paper [CC97] we have not found any reference to the multiple elections issue.

As seen in this analysis, the few implementations of electronic voting

systems do not take in a proper consideration some essential properties for the real use of the system, such as availability, resume-ability and collusion-resistance. These aspects that together we call robustness are the ones that we will tackle in REVS.

Chapter 5

REVS

The availability, resume-ability and collusion-resistance of voting systems are critical properties when considering their deployment in real-life environments. However, in Chapter 4 we noticed that they are often forgotten in the design and in the implementation of voting systems.

The availability of an electronic voting system is critical because those systems are used only in very restricted time frames. Consequently, the unavailability of a voting system can only be acceptable during very short time periods, such as a few minutes.

The availability of an electronic voting system can be compromised by accidental software and/or hardware crash or by deliberate attacks against the system. In an Internet voting system the availability of the system can be affected by known Internet's security problems, such as: tampering basic services (e.g. DNS name resolution) and using DoS attacks against the participating machines or applications (e.g. flooding attacks).

Despite the relevance of availability to the real exploitation of electronic voting systems, we have not seen much discussion about this issue in the analyzed voting systems. For instance, all systems have at least one singular

point of failure, while the first step to improve the availability of a system is the elimination of all singular points of failure.

As there is no solution that guarantees 100% of availability, we should also handle temporary unavailability scenarios, leading us to the resume-ability of voting processes. In electronic voting system we say that a system has resume-ability if it supports interruptions in the voting protocol. Such interruption can be unintentional, when resulting from availability problems, or can be intentional, when the voter decides by its own to stop the voting process to resume it later.

However, to have resume-ability we have to keep some per-voter state of the voting protocol. This state will allow the voter to resume the voting protocol but it also could be used as a receipt. We will come back to this issue in more detail while presenting the REVS voting protocol in Section 5.2.

The remaining aspect of a robust electronic voting system is the collusion-resistance. Inside threats are many times forgotten while trying to protect a system from the outside threats. The inside threats of an electronic voting system result from misbehaviour of one or more colluding electoral authorities with consequences in the properties of electronic voting processes and outcome (cf. Section 4.2.5). Therefore a robust electronic voting system must also minimize the possibility of collusion. Like in paper-based voting processes, the basic principle is divide to conquer: divide the power among several electoral authorities in order to assume that only a large enough set of them is capable to corrupt the voting process.

In this Chapter we present REVS, a Robust Electronic Voting System. Our main goal with REVS was to provide a robust system in all three aspects: (1) availability, by providing a system with no singular points of fail-

ure, and with a voting protocol that supports communication and machine failures; (2) resume-ability, by allowing voters to stop the voting protocol, intentionally or not, and resume it anytime and anywhere latter; and (3) collusion-resistance, by not letting a server alone, or up to a certain configurable degree of collusion, to interfere with the election.

We chose to use a blind signature based voting protocol for REVS. Consequently, as all systems based on blind signatures REVS has some interesting intrinsic advantages, such as: simplicity, low costs (it is computational and network efficient) and ballot independence. These are aspects that are important to facilitate the deployment of REVS to support large scale elections or several types of elections or opinion surveys.

Since an electronic voting system is a complex system, we decided to start from an already existing system and to adapt it to reach our goals. From all the systems analyzed, the Evox-MA is the only one that addresses some robustness issues. However, the results are not as good as they could be (in part because of the weak authentication mechanism used, cf. 4.2.3). Nevertheless we liked the architecture of the system and we used it as the starting point for designing REVS architecture. Naturally, during the description and analysis of REVS along this chapter we will compare it against its direct or indirect predecessors: Evox-MA and Evox.

5.1 REVS architecture

In REVS we have four types of servers: *Ballot Distributor*, *Administrator*, *Anonymizer* and *Counter*. There is also a *Voter Module* that is used by voters to support their participation in elections, and a module to prepare the election called *Commissioner*.

- **Commissioner:** The *Commissioner* is the module used to prepare the election: register the voters, defining the operational configuration for the election (election key pair, ballot, addresses and public keys of servers, number of required signatures, etc.). The *Commissioner* also signs all the election data, so that anyone can verify the authenticity of it. The *Commissioner* can be seen as a kind of electoral commission.

- **Ballot Distributor:** The *Ballot Distributor* is responsible for the distribution, for the voters, of the data set up by the *Commissioner* for the election: ballots and operational configuration. This procedure is expensive in terms of data exchange, so we decided to introduce this dedicated server in REVS. With the introduction of a *Ballot Distributor* we also separated every logical function into a different server, leading to a more modular system than Evox or Evox-MA.

  All the information distributed by a *Ballot Distributor* must be signed by the *Commissioner*, in which all voters trust. Thus, there may be several *Ballot Distributors*. This replication, besides reducing the work load on each *Ballot Distributor*, improves efficiency in large-scale elections and provides toleration to communication or machine failures affecting *Ballot Distributors*, therefore bringing robustness to the distribution process.

- **Administrators:** The *Administrators* are the electoral entities that have the power to decide upon the acceptability of a ballot from a voter. A ballot is acceptable for the final tally of the election only if it has a minimum set of signatures from different *Administrators*. If n is the total number of *Administrators*, a voter must get $t > n/2$ valid signatures from different *Administrators* to make its ballot acceptable.

With such a value for t it is impossible for any voter to get two valid votes.

A voter uses a different password to authenticate himself with each *Administrator*. And because one *Administrator* cannot derive any other password of the voter from the one it knows, as we will show in Section 5.3.3, it cannot alone impersonate a voter.

- **Anonymizer:** The *Anonymizer* server provides anonymity to the voter's machine, preventing a *Counter* from associating a ballot with a machine. The *Anonymizer* hides the voter's location and randomly delays and shuffles several submitted ballots before sending them to *Counters*. The randomization of ballot submissions prevents time analysis trying to associate votes to voters using the time of the participation on the election.

- **Counter:** The *Counter* is the server who verifies the validity of the ballots, checking that all required signatures are on the ballot. Then the *Counter* removes the repeated ballots verifying a bit commitment (made by the voter in the ballot signing phase, see Section 5.2) and performs the tally.

  Voters send their final ballots to the *Counters* through the *Anonymizers*, encrypted with the public key of the election, thus preventing *Anonymizers* and *Counters* from watching votes during the election. REVS can also be used without *Anonymizers*, in this case the voters send their votes directly to the *Counters*; further discussion about the REVS configurations will be made later in this Section.

- **Voter Module:** The *Voter Module* is the module used by the voter to participate in the election. It performs all the proper interactions

with election servers in behalf of the voter, such as: get the ballot, get it signed by *Administrators* and submit it. It also generates the passwords for the voter (see Section 5.3.3) and controls the correct fulfilment of the ballot.

In REVS the *Ballot Distributor* and the *Anonymizer/Counter* pair can be replicated at will. The only server that requires special attention when replicating is the *Administrator*, because having more *Administrators* means more signatures to be obtained by the voters (because $t > n/2$).

REVS can be used with or without *Anonymizers*. As we said before the *Anonymizer* provides anonymity to the machine used by the voter. However there are operational scenarios where this kind of anonymity is unnecessary, for instance: (1) when voting from voting polls, where several voters use the same machines; (2) when voting from home or office using an Internet Service Provider that masquerades the network address; or (3) when using the system in less privacy-demanding environment such as surveys.

5.2 REVS protocol

The flexibility of REVS arquitecture requires a flexible voting protocol. The only restriction made is to the number of required signatures to make a ballot valid, t, which must be greater than $n/2$, where n is the number of *Administrators*.

From the voters' point of view, the REVS protocol is divided in three steps (see Figure 5.1, detailed messages in Figure 5.2):

1) Ballot distribution: The voter contacts a *Ballot Distributor* to get a blank ballot for a given election. The *Ballot Distributor* returns the requested ballot, the election's public key and the operational configuration

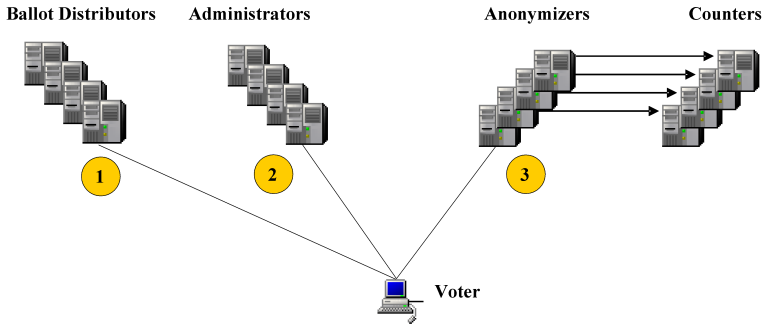

Figure 5.1: REVS voting protocol

of the election, all signed by the election *Commissioner*.

This is done in two phases. First the voter contacts a *Ballot Distributor* and provides a voter ID to receive the list of elections in which he can participate. Then the voter chooses the election and requests a ballot for it from a *Ballot Distributor*.

2) Ballot signing: After expressing his will on the ballot, the voter commits to the ballot digest with a random bit string and blinds the committed digest with a random blinding factor.

Then the voter sends the blinded committed digest of his vote to at least t *Administrators* for signing. The *Administrator*, after receiving a request for signing, verifies if it had already signed for the requesting voter. If not, he signs and saves the signature; if he had signed before, the *Administrator* returns the previously saved data, i.e. the signature of the blinded committed ballot digest. After receiving a signature the voter updates it using an unblinding factor and verifies the correctness of the result using the original ballot digest and the *Administrator's* public key. This process is repeated until all required t signatures are collected.

The voter's module can save the voter's answers, the bit commitment and the blinding factor into non-volatile storage, preferably provided by a

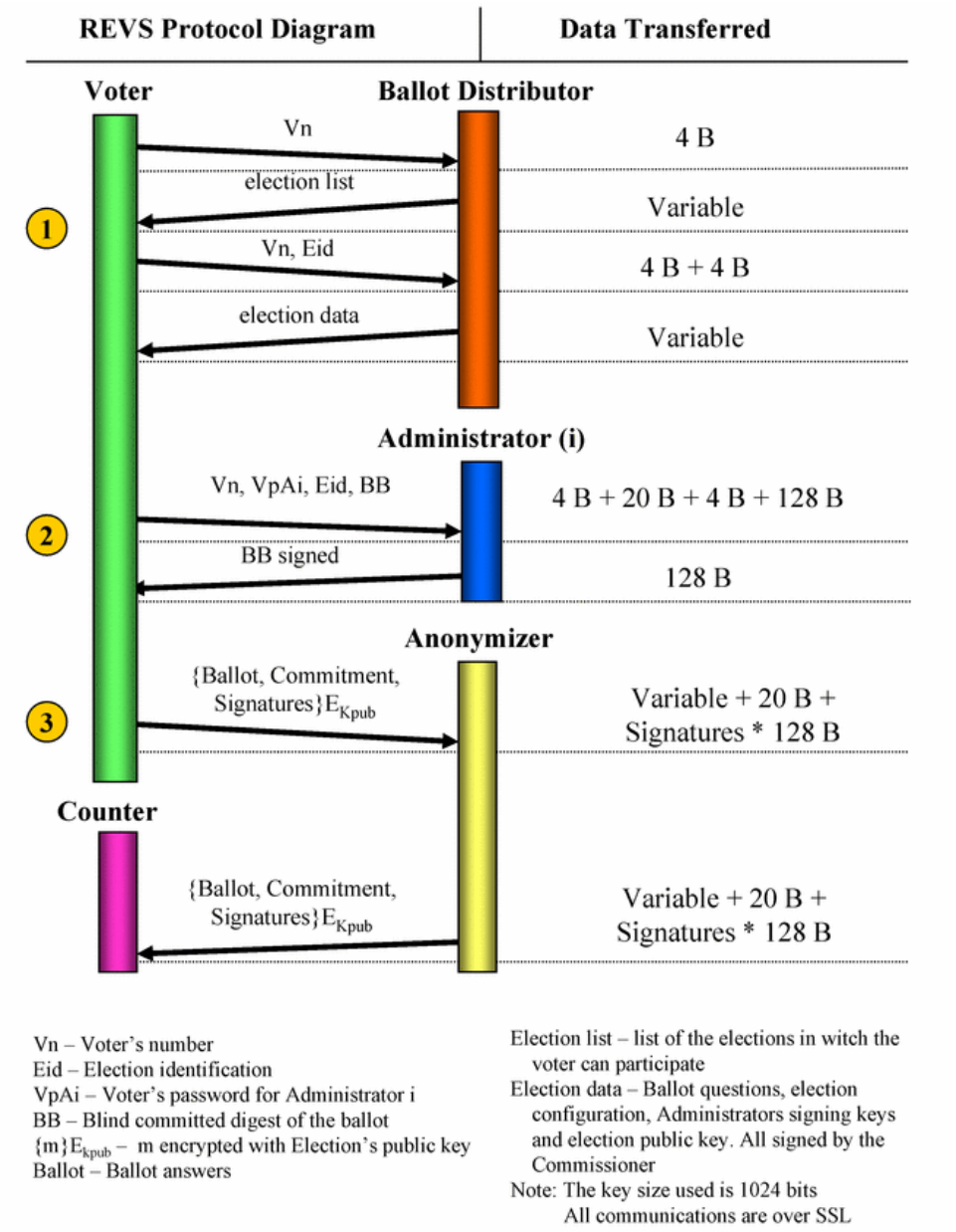

Figure 5.2: REVS voting protocol (message details)

mobile media, before using them. This enables the voter to stop and latter resume its participation in the election, but can affect the voter's privacy because it can be used as a receipt. We discuss this issue at the end of this Section.

3) Ballot submission: in this step the voter constructs the ballot submission package, joining the ballot, its signatures and the bit commitment. At this time the voter can save this data into secure storage. Once again this is an optional step, because it helps improving accuracy but affects privacy (see the discussion at the end of this Section). Then he submits this package, ciphered with a hybrid cryptosystem using a random symmetric session key and the election public key, through an *Anonymizer*, concluding the voting protocol.

The voter can submit the same package to any *Counter* as many times as he feels necessary to be sure that the ballot reaches its destination. This means that different *Counters* can get different sets of votes at the end of the election, and those sets may even contain repeated votes. A selected master *Counter* obtains the final tally after gathering all the valid votes from the several *Counters* and discarding the repeated ones. Any person with access to the ballots collected by all *Counters* can act as a master *Counter*. This fact increases the confidence in the election outcome.

After collecting all votes the counting process involves the following steps:

I. Decipher the submission packages with the election's private key.

II. Verifying that all required t signatures from *Administrators* are present.

III. Removing repeated votes, which are the ones with the same bit commitment. If the length of the bit commitment is large enough (160 bits in REVS) the danger of collisions is negligible.

IV. Tallying the remaining votes.

V. When using multiple *Counters*, the master *Counter* collects all previously verified votes. Then checks for repeated votes using the bit commitment and proceeds with the final tally.

All the *Counters* publish the contents of all received submission packages, and the *Administrators* publish all the signatures provided for the blinded digests. After this publication the voter can verify if his vote was counted. If the vote is not present at the tally he can reclaim presenting, anonymously, the previously saved vote.

5.2.1 Multiple election support

As we describe in Section 4.2.5, none of the analysed systems has a good support for multiple, simultaneous elections. We addressed this problem and REVS allows voters to participate in several elections simultaneously.

REVS allows the use of a different set of signature keys for each election. Such sets prevent the exchange of ballots by the voters, something that could not be done by just controlling the distribution of the ballots. In REVS every *Administrator* has a different asymmetric key pair for each election, so when the voter requests the signature from the *Administrator* he also sends the election identifier. The *Administrator* checks if the voter is registered in the election, and only in that case the *Administrator* signs with the corresponding key. If voters manage to exchange ballots the signatures will not match and the ballots will be invalid, therefore discarded.

5.2.2 To keep or not to keep voting state?

By saving his voting state a voter is able to protect himself against any problem occurring at some stage in the voting protocol. However, this can affect the voter's privacy because the saved state can be used as a receipt.

In fact, the bit commitment used by the voter is used by *Counters* to detect replicas of the same vote, thus the bit commitment is a direct link between a voter and his vote.

Of course, the implications of having a receipt are not the same if we are participating in a political election or if we are answering to an opinion survey. Furthermore, even when participating in a political election, there may be operational scenarios where such receipt is not an issue. For instance, when voting in a voting poll, the voter could save his voting state to allow the resolution of any problem occurred during its voting process and, at the end, the voter could be forced to destroy his voting state before exiting voting poll (to protect his privacy). Or, alternatively, the voter could be forced to put a hardware token containing the state and the final vote in a physical ballot box, in order to protect its privacy and allow an alternative audition of electoral results.

Concluding, REVS simply allows voters to save their voting state for enabling them to stop and latter resume their voting process. Choosing the right way to manage such state for protecting the voters' privacy and to allow to check the results and also to protest against them are political and operational issues beyond the design of REVS.

5.3 Implementation

REVS was fully implemented in Java, enabling it to be installed and executed on any computational platform. We also used a free database back-end in REVS servers, namely the 3.23.53-max version of MySQL. To provide secure communication channels between the voter and the servers we use JAVA RMI with SSL/TLS. Using SSL/TLS channels also allows an authentication of the servers using their asymmetric key pair. More details about the im-

plementation are presented in the following sections. The steps needed to install, configure and run REVS are presented in Appendix A.

5.3.1 REVS modules and servers

Servers

All servers have three layers: 1) server interface, 2) server engine and, 3) server database. In the server interface are defined the services provided by the server. The server engine is where all the logic of the server is implemented. The methods to access permanent data storage are implemented in the server database layer. All services provided are idempotent, with also idempotent database accesses, therefore, for larger elections, a cluster can be easily implemented to improve performance of the servers.

Ballot Distributor The Ballot Distributor server provides two services: the *getElectionsList* and the *getBallot*. The first service returns a list of elections in which the voter can participate; the second service returns the ballot and all the other needed information to participate in the election (the election public key, the number of required signatures, the administrators' public keys, etc.). All services provided by this server are stateless, therefore there is no restriction to the replication of this server.

Administrator The Administrator server only provides the *sign* service. The first time the voter requests the service the administrator signs the blind ballot and stores the signature in the database; in subsequent requests it is returned the signature made for the first request. Since the storage of the signature in the database is atomic and the signing process is computationally heavy, there could be used a cluster, sharing

a database, to provide more computational power.

Anonymizer The Anonynizer hides the network address of the voter machine. The REVS Anonymizer has a thread that runs from time to time (random interval). It verifies if there are some votes waiting to be forwarded to the Counter, if so a random number of them are forwarded to the Counter. Since the use of the Anonymizer is optional (cf. Section 5.1) we defined the interface *ISubmission*, where it is defined the vote submission service, *submitVote*, and both Anonymizers and Counters implement this interface. Since a primary goal of REVS is robustness, we use this simple way to implement an anonymous channel. However, more sufisticated anonymous channel can be used with REVS, they only need to implement the *ISubmission* interface.

Counter As said before, the Counter server implements the *ISubmission* interface to allow transparency in the vote submission process. The Counter additionally provides the *getVotes* service; this service is used at the end of the election, by a master Counter, to collect all the votes received by the Counter. The Counter also has the tasks of deciphering and verification of the votes. To decipher the votes it is necessary to use the election's private key, something that is only possible with the cooperation of the electoral commission (because the key is in its hands, see the Commissioner description bellow).

Voter Module

The Voter Module is the voter interface to REVS, it performs all the steps and cryptographic transformations needed by the voting protocol. This module is also responsible for displaying the ballot and for collecting the

voter's answers. The Voter Module has a modular design divided in three parts: (1) the voter's interface, implemented by the *FrameVoter* class; (2) the *VoterEngine* class, containing all the logic of the module; and (3) an *IBallotViewer* implementation, to display the ballot and collect the answers.

The Voter's Module can be distributed to the voters in the registration process or can be downloaded later from the election official site or from some Ballot Distributor server.

Commissioner

The Commissioner server is the server used by the electoral commission to prepare the election. It is used to register the voters, to define the election configuration, and to create the elections' asymmetric key pairs. The elections' private keys are encrypted and saved in a file, the key used to encrypt the file is signed by the Commissioner and also saved in a file. These two files should be kept secret and appart until the end of the election by the electoral commission. Anyone with access to the two files can, in collaboration with a Counter produce intermediate results, therefore in a future release the elections' keys should be created in a threshold way (cf. Section 3.2.1).

5.3.2 Cryptography

In REVS all signatures are 1024-bit RSA-SHA1 signatures, i.e., RSA signatures, using 1024-bit keys, over the digest of the data made with the SHA-1 algorithm. For symmetric cipher it was used the Triple-DES algorithm. For ciphering the ballot submission package with an hybrid cryptosystem it was used the RSA-Triple-DES combination: data ciphered with Triple-DES and the symmetric key ciphered with RSA.

To perform blind signatures we developed the *RSABlindSignature* class,

as the name suggest it implements RSA blind signatures, cf. Section 3.3.1.

5.3.3 Voter's authentication

For authenticating voters we used the well-known username/password method. The voter must use a different password for each different *Administrator* for preventing impersonation. But, for keeping the authentication user-friendly, we should not force the voter to memorize several passwords. Thus, we designed an algorithm for generating all necessary passwords from a small set of secrets.

Our algorithm uses two secrets (see Figure 5.3): a strong password (non trivial password, like a large random alphabetical string, that could be keep safe inside an envelope or in a magnetic card or even in a tamperproof smartcard) and an activation PIN, that should be memorized by the voter. The voter introduces these two passwords in the *Voters Module* and it computes all required passwords, one for each *Administrator*. Because the algorithm uses digest functions, an *Administrator* Ai knowing a voter's password Pi cannot compute any other password Pj, $j \neq i$, known by *Administrator* Aj. With this algorithm we prevent the impersonation of voters by less than t colluded *Administrators*.

For improving security, the voter should choose both passwords. At a registration phase the voter should give the actual passwords, in an Unix like method, that will be used to authenticate itself to the *Administrators*. Another possibility is to give the passwords to the voters and allow them to choose new ones.

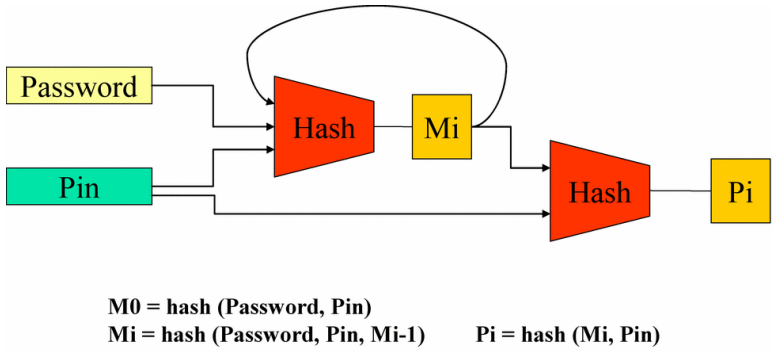

Figure 5.3: Password generation algorithm

5.3.4 Ballots

One of the advantages of the blind signature protocols is ballot independence. Taking advantage of this property, we have implemented the ballots and answers in XML. This way we bring several others advantages to the system, such as:

- It is easy to build ballots; it can be done in any text editor.
- Anyone who knows a little of XML can easily read the ballots and answers.
- It easy to extend the ballot, e.g. add new question types, add additional information, etc.

In Figure 5.4 is presented the general XML construction of ballots and answers. A ballot is composed by a description and several groups of questions. A group of questions has a description and several questions. A question is composed by a description, the question it self, and by the possible answers.

Currently four types of questions are supported:

- **Single:** the answer must be one and only one of the presented choices;

```
<!-- General ballot example -->

<ballot electionCode="123">
    <ballotDescription>
        <line>Test Ballot description line one</line>
        <!-- more lines -->
    </ballotDescription>
    <group code="1" description="Group one">
        <!--type tag can also have the values Multiple, OpenS or OpenM->
        <question code="1" type="Single">
            <questionDescription>First Question</questionDescription>
            <answer code="1">First Answer</answer>
            <!-- more answers -->
        </question>
        <!-- more questions -->
    </group>
    <!-- more groups -->
</ballot>
```

```
<!-- General ballot answer example -->

<ballotAnswer electionCode="123">
  <group code="1">
    <!-- Single  type -->
    <question code="1" answer="1"></question>
    <!-- Multiple  type -->
    <question code="2" answer="1-3-7"></question>
    <!– OpenS  type (selecting the open answer)-->
    <question code="3" answer="0">Open answer</question>
    <!– OpenS  type (selecting another answer )-->
    <question code="4" answer="5"></question>
    <!– OpenM  type -->
    <question code="5" answer="0-2-4">Open answer</question>
    <!-- more answers -->
  </group>
  <!-- more groups -->
</ballotAnswer>
```

Figure 5.4: XML ballot and answer

- **Multiple:** we can choose any number of choices for our answer;
- **OpenS:** open single;
- **OpenM:** open multiple.

The OpenS and OpenM types are similar to the Single and Multiple types respectively, but it is also possible to give another answer. The OpenS and OpenM types of question are very useful for surveys. Because all questions have a well defined type the *Voter Module* can verify the correct fulfilment of the ballot.

5.4 Evaluation

The evaluation of the functionality of REVS is made under several assumptions. First we will clarify those assumptions, and then evaluate REVS considering the properties presented in Section 2.2. The assumptions are:

- The cryptographic algorithms used are hard to break. REVS uses three cryptographic algorithms: (i) RSA, for producing and checking

blind and non-blind signatures, and also for encrypting the keys used to encrypt the submission packages; (ii) Triple-DES, used to encrypt the submission packages; and (iii) SHA-1, for all required digest computations.

- The communications are secure. All communications in REVS use SSL and servers are authenticated with their public keys witch are certified by the *Commissioner*.
- The voters' computers are clear from infection by Trojan horses or viruses.
- The required number of signatures respects $t > n/2$, and there are t honest *Administrators*.
- The voter decides to save his data in secure non-volatile storage in steps I and II of the voting protocol.

Accuracy

A vote cannot be altered because that would invalidate all *Administrators'* signatures. A voter can verify if his vote was eliminated from the final tally, by examining the list of received votes published by *Counters*, and can correct this by sending his submission package anonymously. The elimination of votes when using several *Counters* is not a trivial task because it implies the elimination of the vote from all *Counters*; even so the voter can anonymously submit his ballot and correct the tally. Because the signatures can be verified by anyone and are published with the votes, it is impossible for an invalid vote to be part of the final tally. Therefore, all three aspects of accuracy are respected.

Democracy

Each voter can only vote once in each election because $t > n/2$ (a voter cannot produce two valid votes). To prevent a voter from voting it is needed the collusion of $n - t + 1$ *Administrators*; the voter is prevented from voting if he is prevented from obtaining the t required signatures.

In REVS the introduction of false valid votes is only possible if there is t colluding *Administrators*. As in the other systems analyzed, this fraud can be detected but cannot be corrected. However, in the case of introduction of votes for absentee voters, the fraud can pass undetected.

The leak of partial results in REVS is only possible with the collusion of the electoral commission and a *Counter* or an *Anonymizer*. The *Counter* or *Anonymizer* have the ciphered votes and the electoral commission the election private key.

Concluding, if the electoral commission and at least t *Administrators* are honest all aspects of democracy are guaranteed.

Privacy

While the *Anonymizers* are honest no electoral authority can link a vote to a voter, the only way to break this anonymity is by a collusion of the *Administrator*, *Anonymizer* and *Counter* servers.

As all other voting prototypes analysed, REVS is not receipt-free, therefore the second part of the privacy property in no accomplished.

Verifiability

The final tally can be made by anyone by verifying the signatures on the votes and summing all votes. Each voter can verify if its own vote is correct, and assumes that the other votes are correct because of the signatures they

have.

Robustness

- **Availability -** Since all servers can be replicated REVS has no single point of failure. The system is available as long as there is a minimal set of servers running correctly. The minimal set is actually one *Ballot Distributor*, t *Administrators*, and one *Anonymizer* or *Counter*.

- **Resume-ability -** As explained in Section 2.2, the voter can recover from an interruption in the voting protocol as long as the voter keeps its voting data, *i.e.* the bit commitment and the blinding factor.

- **Collusion-resistance -** In REVS no electoral authority alone can disturb any property of the system. However the collusion against the democracy property deserves special attention. To affect the democracy property, in a configuration with n *Administrators* and t required signatures, it needs the cooperation of t *Administrators* to cast a valid vote (increases as t grows). To prevent a voter from voting $n - t + 1$ *Administrators* must conspire, preventing the voter from obtaining the required t signatures (decreases as t grows). So, its obvious that its necessary to make a trade-off between these two opposite weaknesses. Furthermore, to increase fault-tolerance t should be as low as possible.

		Sensus	EVOX	EVOX-MA	Vienna's voting system	REVS
Authentication		PKI	Username/ Password	Username/ Password	PKI	Username/ Password
Accuracy	Elimination of votes	Detect and correct	Detect and correct	Detect and correct	Detect and correct	Detect and correct
	Introduction of invalid votes	Detect and correct	Detect and correct	Detect and correct	Detect and correct	Detect and correct
	Change vote	Invalidate vote	Invalidate vote	Invalidate vote	Possible with no correction	Invalidate vote
Democracy	Prevent voter from voting	Detect and correct	Administrator	Manager or *(n-t* +1) Administrators	Detect and correct	*(n-t* +1) Administrators
	Introduction of valid votes	Administrator	Administrator	Manager and $t \leq$ Administrators	Registration and Trust Center servers	*t* Administrators
	Voting for absentee voters	No	Administrator	Manager and *t* Administrators	No	*t* Administrators
	Leak of partial results	Counter	Counter and Anonymizer	Counter and Anonymizer	Counter	Electoral commission, Counter and/or Anonymizer
Privacy	Anonymous channel	Need	Uses Anonymizer	Uses Anonymizer	Need if the voter uses same PC	Uses Anonymizer
	Collusion to break anonymous channel	Administrator and Counter	Administrator, Anonymizer and Counter	Administrator or Manager, Anonymizer and Counter	Registration or Trust Center and Counter	Administrator, Anonymizer and Counter
	Receipt-free	No	No	No	?	No
Verifiability	Own vote	Yes	Yes	Yes	Yes	Yes
	All votes	Signatures verification	Signatures verification	Signatures verification	Signatures verification?	Signatures verification
Robustness	Availability	Single points of failure (all servers)	Single points of failure (all servers)	Single points of failure (except Administrators)	Single points of failure (all servers)	No single points of failure
	Resume-ability	No	No	Only for *n-t* Administrators	Yes	Yes
	Collusion-resistance (accuracy)	Yes	Yes	Yes	No	Yes
	Collusion-resistance (democracy)	No	No	No	No	Yes
	Collusion-resistance (privacy)	2 servers	3 servers	3 servers	2 servers	3 servers

Table 5.1: Overall systems analysis with REVS

Table 5.1 presents a side-by-side evaluation of the systems presented in Section 4.2 and REVS. In this table we can observe that the goals of REVS were accomplished, REVS robustness is from far the best without compromising the other fundamental voting properties.

5.4.1 Performance evaluation

REVS was designed to support large scale elections. In this section we evaluate the prototype of REVS concerning implementation decisions, time consumed in cryptographic functions, both by the voters module and by *Administrators*, and amount of data transferred.

As seen before, REVS can run without single points of failure, therefore avoiding bottlenecks. All the servers have a database back-end and were implemented in a way that, if necessary, a cluster can be easily implemented to improve performance. With this design and implementation considerations we believe that REVS can easily support large scale elections.

To evaluate the performance of the prototype we have made some tests using a computer with a Pentium III processor at 550 MHz, 384 MB of RAM running Windows XP Professional.

Regarding the *Voter's Module* we determined that it would take less than half a second to compute 1000 passwords; the blinding process is done in less than 200 ms; and the verification of a blind signatures is done in less than 30 ms. So, when using REVS in a configuration requiring 5 signatures the *Voter's Module* would compute all cryptographic operations in about a second.

The *Administrator* is the other entity that must compute cryptographic operations (signing blinded digests of the ballots). In the tested machine we verified that an *Administrator* takes less than 200 ms to verify the voter identity and sign the blinded ballot (about 15% for the first action and 85% for the second). An *Administrator*, with the test configuration, can produce 5 signatures per second. For getting better performance results we can use a more powerful computer or deploy the *Administrator* as a cluster of machines sharing a high-performance database.

	Ballot Distribution	Ballot Signing	Ballot Submission	Total (per voter)
n=3 t=2	≈ 3.3 KB	≈ 0.5 KB	≈ 1.1 KB	≈ 4.9 KB
n=5 t=3	≈ 3.4 KB	≈ 0.8 KB	≈ 1.3 KB	≈ 5.5 KB
n=7 t=4	≈ 3.5 KB	≈ 1 Kb	≈ 1.5 KB	≈ 6.0 KB

Table 5.2: Protocol data transfer resume

Besides servers' performance, its also necessary to analyze the amount of data transferred in the protocol (cf. Figure 5.2). Being REVS a ballot independent voting system we must make some assumption before analyzing any data transfers: we assumed that the elections' list is 1 KB long; the election data is divided in a fixed part with 2 KB (ballot questions, election configuration and election public key), and a variable part with n * 1024 bits long (public keys of the *Administrators* running the election), finally we assume that the vote (the ballot answers) is 256 bytes long.

From the figures presented in Table 5.2 we can conclude that REVS offers a good data transfer performance. For instance we can compare the data transferred for each voter with the size of the Google's main page, which is about 16 KB. REVS also provides a good trade-off between increased security and data transfer, about 0.5 KB for each additional signature required. Note that all calculations do not take into account the additional traffic generated by communicating over SSL.

Another important aspect to observe from Table 5.2 is that 60% to 75% of all data transferred is relative to the Ballot Distribution phase. Therefore it was a good option to use stateless dedicated servers for this operation.

Regarding the required computation and data transfer aspects, the pre-

viously presented figures allows us to conclude that our prototype of REVS is efficient and can be used in large scale elections.

5.5 The first experiment

The first prototype and experiment of REVS was done in the Instituto Superior Técnico to support elections, namely surveys for evaluating the quality of courses. To this particular scenario, REVS servers were deployed and managed by separate entities, namely central computer services, several departments and students organizations, in order to reduce the possibility of collusion. A set of trusted machines was made available to voters, but they could use any other machine to participate in the elections.

The students were able to vote during a period of two weeks. During the survey period it was achieved 100% of availability, even in the presence of a few server and communications failures. This first experiment proved the robustness of REVS.

5.6 Proposed improvements

In 2004 Lebre *et al.* [LJZF04] proposed two changes in the voting protocol to make it even more robust. The first change addresses the problem of malicious *Administrators* trying to prevent voters from voting by not signing correctly the voter's ballot. The second change tries to prevent the submission of garbage to the *Counter*: the data submitted to the *Counter* is encrypted, therefore there is no way to distinguish a real encrypted vote from a garbage. We will now present a resume of the improvements proposed.

Both improvements use a certified key pair generated by each voter during the voting process (KVpub, KVpri). The voter certifies his key KVpub with

the signature of k *Administrators*. This process can be repeated as many times as the voter wants with any KVpub. After acquiring the required signatures, the protocol is similar to REVS. The voter sends the blinded digest of the committed ballot for signing, but now signs it with KVpriv. In the same message goes the KVpub certificate, i.e. the key KVpub and its k *Administrators'* signatures. Now each *Administrator* only signs a voter's blinded digest if all its k signatures are correct, and any supervision authority can verify the correctness of such decision. Likewise, an incorrect reply from an *Administrator* can be detected by a supervision authority. This is repeated until all the required t *Administrators'* signatures are obtained. The *Administrators* store the complete voters' requests as a proof for future claims. Note that in both interactions, to certify the key pair and to obtain the signature on the blinded ballot, the voter authenticates himself to each *Administrator* contacted with the current username/password scheme of REVS.

The solution proposed for the second problem is similar to the one previously used for signing voters' requests sent to *Administrators*. A submission package is signed by the voter's private key and the corresponding certified public key is sent along with the submission package. Since only authorized voters can get the required blind certificate, only authorized voters may successfully interact with *Anonymizer/Counters*. The key pair used for authenticating submitted votes must be different from any of the key pairs used for authenticating voter's requests to *Administrators*. Otherwise *Administrators* could easily link votes to voters by checking the results publicly presented by *Counters* at the end of elections. Note that this solution does not prevent authorized voters from submitting correctly signed random data. But this behavior can be detected by keeping some extra state in *Counters* and us-

ing auditing systems in *Administrators*. If Counters store blind certificates in order to filter out different submissions using the same certificate, then malicious voters are forced to require several blind certificates from *Administrators*. This way they reveal their potentially malicious attempts and can easily be identified by election authorities, because the interaction with *Administrators* requires identification of voters. Another possibility is to allow *Administrators*, under a proper supervision, to slow down blind certifications for voters requiring them very often.

Chapter 6

Conclusions and future work

The robustness of voting systems is essential for our democracies. The democracy principle is the right that everyone has to vote; without a robust voting system people could be prevented to vote, therefore corrupting the democracy basic requisite.

A robust electronic voting system has to have fault-tolerance, to communications or servers failures, and also should have protection against the misbehaviour of isolated or colluded electoral authorities. However, that is not what we observe in the analysed implementations of electronic voting systems, in part because an Internet voting system is a complex application and any slight change in it can seriously compromise the system.

REVS goal was to provide a robust system concerning three aspects: (1) availability, by providing a system with no singular points of failure, and with a voting protocol that supports communication and machine failures; (2) resume-ability, by allowing voters to stop the voting protocol, intentionally or not, and resume it anytime and anywhere latter; and (3) collusion-resistance, by not letting a server alone, or up to a certain configurable degree of collusion, to interfere with the election. The REVS goal was fully

accomplished, as shown in Section 5.4.

REVS is also a very configurable system (cf. Section 5.1) therefore easy to adapt to distinct operational environments, either opinion surveys or national elections.

REVS has also all the advantages of being a blind signature based voting system, such as: simplicity, low costs (it is computational and network efficient) and ballot independence. Taking advantage of the ballot independency we defined the ballots an answers in XML. Therefore, it is easy to expand the ballot format to accommodate future needs.

Concerning the multiple elections support REVS is the only, from the analysed systems, that allows the secure realization of simultaneous elections without electorate restrictions, therefore the only one that provides a real multiple elections support.

Concluding, REVS is a very configurable Robust Electronic Voting System that can be easily used to support a large variety of elections and surveys in a secure and robust way.

Future work

To facilitate the election preparation and analysis there should be developed some add-ons to REVS, such as: a graphic ballot editor and a graphic results analyzer.

The generation of the election key pair should be reviewed to give a better protection against the leak of partial results, in future releases of REVS there should be an option to create and use threshold election keys.

Another aspect that should also be reviewed is the anonymous channel, we provide a simple anonymous channel construction based on the use of an *Anonymizer* server. However, as explained in Section 5.1, the need to

use an anonymous channel depends of the operational environment of the election/survey.

REVS uses a username/password authentication mechanism because it is more user-friendly to the general public. However, we are aware of the security advantages of the public key authentication mechanism, therefore we intend to integrate this type of authentication on REVS.

The mobility of voters is the major contribution that electronic voting systems can give to the democratic voting process. With the developments of the mobile communications world it starts to become possible to extend the mobility of voters to allow voting virtually from anywhere anytime. We plan to develop a *Voter's Module* of REVS for the mobile communications environment. The deployment platform of this new *Voter's Module* could be smart-phones or even GSM/UTMS smartcards.

Appendix A

Installation and use

A.1 Pre-requisites

To run REVS it is needed additional software, namely:

- MySQL version $\geq$ 3.23.53 MAX installed
  (available at http://www.mysql.com/)
- Java runtime version $\geq$ 1.4 installed
  (available at http://java.sun.com/)

Optional software

- OpenSSL (available at http://www.openssl.org/)

A.2 Key management

After the installation of the required software, the first step is to create and sign the keys of all servers (Commissioner, Ballot Distributors, Administrators, Anonymizers and Counters). To generate the keys we used the Java command line tool *keytool*.

A.2.1 Create a key

To create a key type the following command should be executed in the command line:

keytool -genkey -keystore kstore.ks -alias server -keyalg RSA -keysize 1024 -validity 365

This command creates a 1024 bits RSA key valid for 365 days. The key is stored in a keystore file named kstore.ks with the server alias. To find out more about keytool utility please check the Java documentation.

A.2.2 Sign a key

First we must create the signature request and submit it to a Certification Authority (CA) for signing. To create a signature request type the following command:

keytool -certreq -keystore kstore.ks -alias server -file server.req

This command will create a signature request for the key with the alias server and store it in the *server.req* file.

The second step is to get the certificate request signed. You can get your keys signed by a certification authority such as VerySign or you can create your own CA. We used the OpenSSL tool to create our own CA. After installing OpenSSL properly we use the following command to sign the request:

openssl x509 -req -in server.req -out server.crt -CA demoCA\cacert.crt -CAkey demoCA\private\cakey.crt -CAserial demoCA\serial

This command will use the CA installed in the *demoCA* directory to sign our request, the *cacert.crt* contains the CA public key certificate and the

cakey.crt contains the CA private key. The signed request is stored in the *server.crt* file.

A.2.3 Import the signed certificate

Before importing the signed public key certificate we should first import the CA public key certificate.

keytool -import -file cacert.crt -keystore kstore.ks -alias ecca

This command imports the CA public key certificate *cacert.crt* to the keystore using the alias ecca (electoral commission certification authority). Then we can import our signed certificate:

keytool -import -file server.crt -keystore kstore.ks -alias server

Since the CA certificate is already in the keystore, it is possible to verify the signature on it and construct a valid certificate chain.

Note: use a different keystore file for each server.

A.3 Installing servers

A.3.1 Configuration file

For the Ballot Distributor, Administrator, Anonymizer and Counter servers there should be a configuration file defining the server and database addresses.

The configuration file is a text file that should look like this:

SERVER <address (//host/service_name)>
DATABASE <address (//host/database)>

Example:

SERVER //localhost/administrator

DATABASE //localhost/adm_ database

A.3.2 Setting up servers

We have separated REVS in two jar files (*revs_ servers.jar* and *revs_ voter.jar*). For setting up the servers we use the *revs_ servers.jar* file. To set up a REVS server follow these steps:

1. Create the server's database in MySQL.
2. Copy the *revs_ servers.jar* to the installation directory.
3. Create the subdirectories *"conf"* and *"ext"*.
4. Copy to the *"conf"* subdirectory the following files:
   (a) *kstore.ks* file: containing the key of the server, the signed public key certificate by the CA and the CA public key certificate (cf. Section A.2).
   (b) *tstore.ks* file: containing the CA public key certificate (only for Anonymizers and Counters).
   (c) *commissioner.crt* file: The commissioner public key certificate signed by the CA.
   (d) *server.cfg* file: the server configuration file (cf. Section A.3.1).
   (e) *policy.txt*: this file is a Java policy file; for more information about it consult the Java documentation. An example of a policy file is available at REVS download site.
5. Copy to the *"ext"* subdirectory the following files:

(a) *soap.jar*: available at REVS download site.

(b) *mysql-connector-java.jar*: available at REVS download site and at MySQL site.

Now we are ready to start the server. To start a Ballot Distributor, Administrator, Anonymizer or Counter server just type the following command:

java -classpath "revs_ server.jar;ext/soap.jar;ext/mysql-connector-java.jar;" -Djava.security.policy=conf/policy.txt -Djava.rmi.server.codebase= file:/<full_ directory_ path>/revs_ servers.jar inescID.revs.servers.StartServer

If everything is ok it should appear a menu to choose the server's type:

Select server type

0 - Distributor

1 - Administrator

2 - Anonymizer

3 - Counter

Server type:

After selecting the server's type it will be asked for the passwords for the database authentication, the keystore and the private key:

Press Enter for defaults.

user: REVSuser

password:REVSpassDB

KeyStore

password: REVSpassKS

Private key

password: REVSpassPK

The default values are only for the database authentication (user: sa, password: *<no_password>*). Finally, there should appear a list of actions allowed by the selected server:

K - Create signing keys (only Administrator)

F - Forward Counter selection (only Anonymizer)

G - Gather votes (only Counter)

T - Tally votes (only Counter)

C - Create database

D - Delete database

R - Redo database

U - Update database

S - Start server

E - Exit

Option:

To start the Commissioner server type the following command:

java -classpath "revs_servers.jar;ext\mysql-connector-java.jar;"
-Djava.security.policy=conf/policy.txt inescID.revs.commissioner.Commissioner

First it will be asked for the authentication information:

Press Enter for defaults.

user: REVSuser

password:REVSpassDB

KeyStore

password: REVSpassKS

Private key

password: REVSpassPK

And then the actions menu should appear:

C - Create tables

D - Delete tables

R - Redo tables

F - Fill tables

G - Graphic mode

E - Exit

Option:

All servers have three database management actions: create, delete and redo. **Before we can start using a server for the first time we must create the database tables.**

The remaining actions of each server will be explained in the next Sections.

A.4 Setting up an election

In REVS the election is prepared by using the *Commissioner* server. To set up an election start the *Commissioner* server as described in Section A.3.2; if it is the first time do not forget to create the database tables. Then choose the option *G* to enter the graphic mode (see Figure A.1), alternatively you can start the *Commissioner* server with the *-G* option (add *-G* at the end of the command to start the server). Now just follow these three steps:

1. First it is necessary to register the voters, option *Voters* in the *Commissioner* main menu (Figure A.1). In the *Voter Administration* menu it is possible to add, remove or change the voters' records (Figure A.2). When defining the passwords of the voters there are two options: a password and a pin or only one password (cf. Figure A.3). In the

case of using only one password the system internally splits it into two pieces, a "password" and a pin, to be used in the authentication algorithm defined in Section 5.3.3.

The voters are organized in groups and each voter can belong to several groups. To manage the groups of voters choose the option *Voters Groups* in the *Voter Administration* menu. In the *Group Administration* menu (Figure A.4) it is possible to add and remove groups, to rename the group and to manage the voters in the groups (Figure A.5). Note that the election electorate will be a voters' group.

2. The second step is to define an election configuration, option *Configurations* in the main menu. To define an election configuration it is necessary to define the polling period (start and end dates), the number of *Administrators* to use, the required signatures to make a vote valid and if *Anonymizers* are to be used. The *Configuration Administration* menu is shown in Figure A.6.

3. To finish the election setups select the option *Elections* in the main menu. In the *Election Administration* menu (Figure A.7) it is possible to create, delete or edit elections. To define an election it is necessary to define the name of the election, the election's ballot (cf. Section A.4.1), the election's electorate (a voters' group) and the election's configuration. Note that several elections can use the same voters' group and/or election configuration.

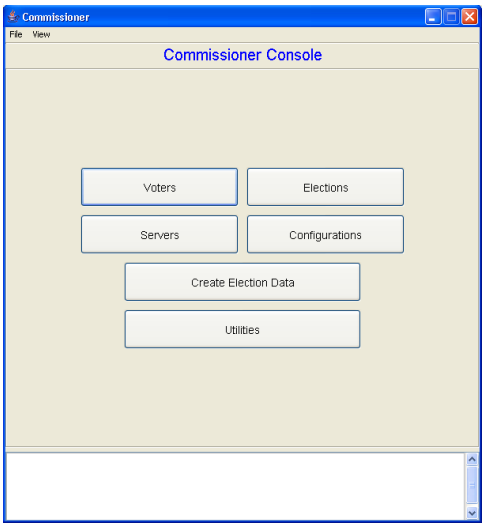

Figure A.1: Commissioner main menu

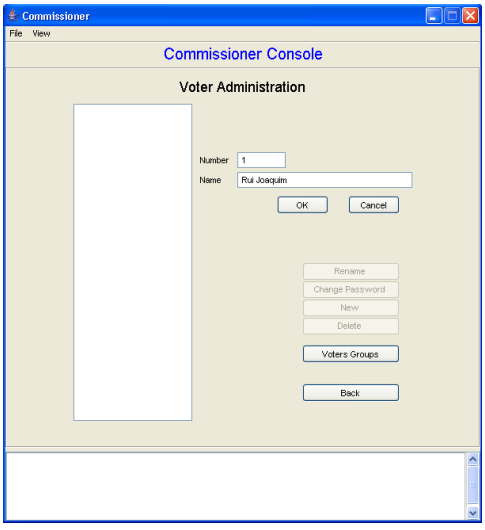

Figure A.2: Voters Administration menu

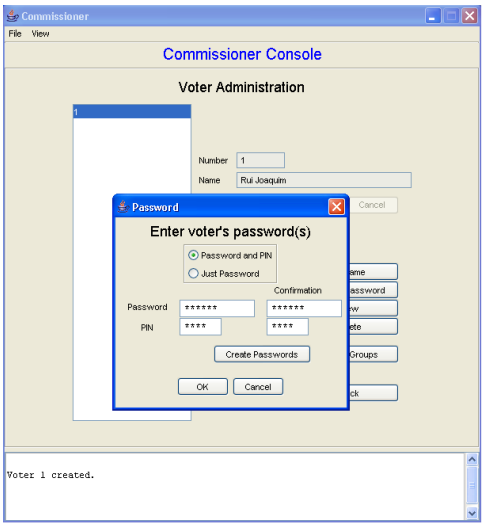

Figure A.3: Password menu

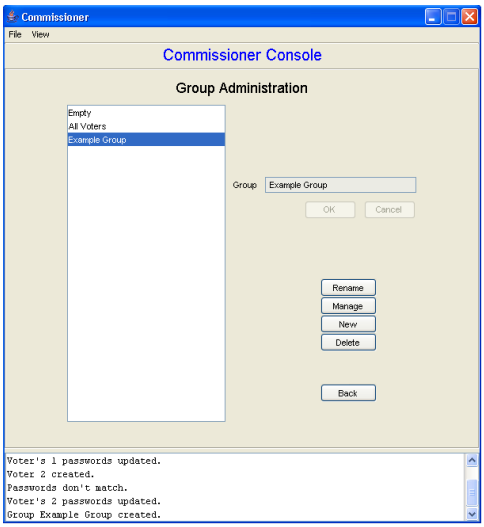

Figure A.4: Group Administration menu

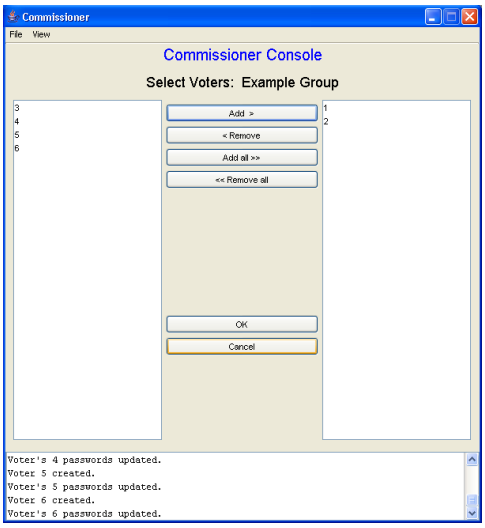

Figure A.5: Group management menu

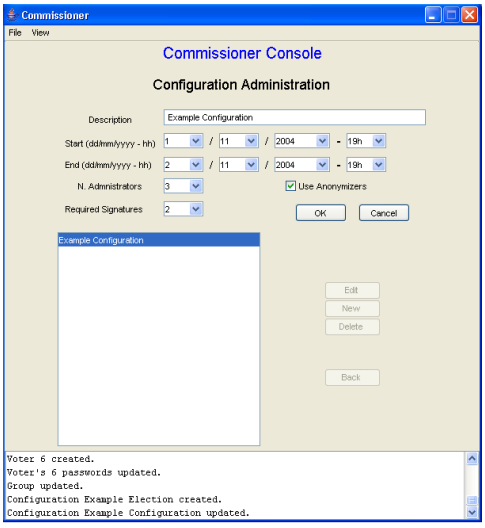

Figure A.6: Configuration Administration menu

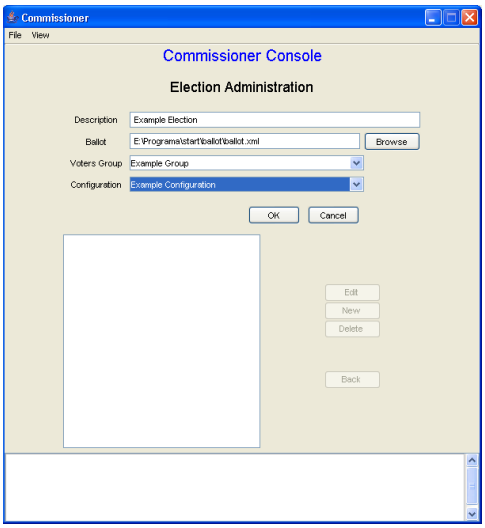

Figure A.7: Election Administration menu

A.4.1 Create a ballot

The ballots are defined in XML as presented in Figure A.8. A ballot is composed by a description and several groups of questions. A group of questions has a description and several questions. A question is composed by a description, the question it self, and by the possible answers.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- General ballot example -->

<ballot electionCode="0">
    <ballotDescription>
        <line>Example ballot</line>
        <line>Extra line</line>
        <!-- more lines -->
    </ballotDescription>
    <group code="1" description="Simple questions types">
        <!--type tag can have the values Single, Multiple, OpenS or OpenM-->
        <question code="1" type="Single">
            <questionDescription>Is REVS robust?</questionDescription>
            <answer code="1">Yes</answer>
            <answer code="2">No</answer>
            <answer code="3">Don't know</answer>
        </question>
        <question code="2" type="Multiple">
            <questionDescription>Do you plan to use REVS in:</questionDescription>
            <answer code="1">National elections</answer>
            <answer code="2">Opinion surveys</answer>
            <answer code="3">Student elections</answer>
            <!-- more answers -->
        </question>
        <!-- more questions -->
    </group>
    <group code="2" description="Open questions types">
        <question code="1" type="OpenS">
            <questionDescription>What is your favorite color?</questionDescription>
            <answer code="1">Red</answer>
            <answer code="3">Green</answer>
            <answer code="4">Yellow</answer>
                                        <!-- more answers -->
        </question>
        <question code="2" type="OpenM">
            <questionDescription>What do you like to do in your free time?</questionDescription>
            <answer code="1">See a movie</answer>
            <answer code="2">Read a book</answer>
            <answer code="3">Go for a walk</answer>
                                        <!-- more answers -->
        </question>
        <!-- more questions -->
    </group>
    <!-- more groups -->
</ballot>
```

Figure A.8: XML ballot

Currently four types of questions are supported: Single, the answer must be one and only one of the presented choices; Multiple, we can choose any number of choices for our answer; OpenS (open single) and OpenM (open multiple) types are similar to the Single and Multiple types respectively, but

it is also possible to give another answer.

Currently there is no specific ballot editor. Therefore, it is necessary to use a text editor to create the election ballot.

A.4.2 Defining the election servers

Part of the setting up of REVS consists in defining the election servers, option *Servers* in the main menu. In the *Servers Administration* menu (Figure A.9) it is possible to define the address and import the public key of the elections servers (Ballot Distributors, Administrators, Anonymizers and Counters). To import the public key of the server load the public key certificate file (cf. Section A.2). Only the servers that are enabled can be used in the election.

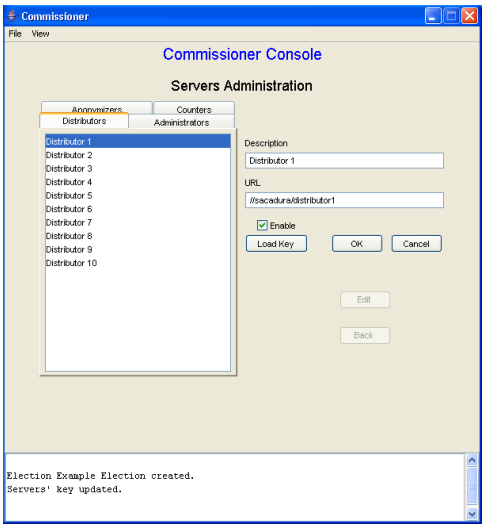

Figure A.9: Servers Administration menu

A.4.3 Import voters and elections

It is possible to import voters and elections from text files, using the appropriate commands at the *Utilities* menu, cf. Figure A.10. The text files should have the following format:

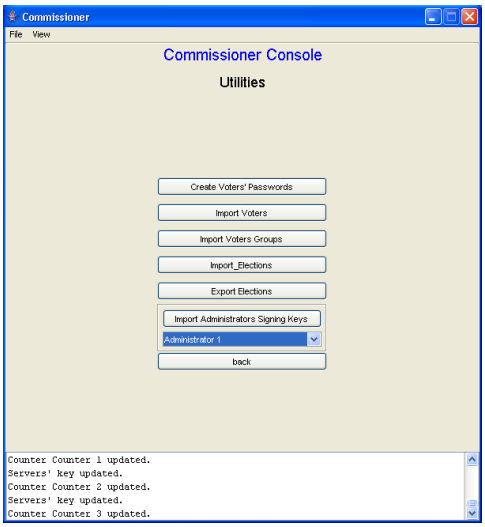

Figure A.10: Utilities menu

- Voters file (one line per voter):

  <id>;<name>[;password[;pin]]

  If the voters have no password information, use the option *Create Voters' Passwords* in the *Utilities* menu to create them.

- Voters' groups (one line per association group->voter):

  (<group id>/<group description>);<id_voter>

- Elections file (one line per election):

  [<election id>;]<election description>;(<voters group id>/<voters group description>);<election configuration description>;<ballot file>

  Note that the election configuration must be created previously to the import of the elections file.

A.5 Start an election

To start an election it is necessary to create the servers' databases, option *Create Election Data* in the main menu of the *Commissioner*. The databases are created based on an election configuration instead of based on individual elections. Therefore, the databases created contain information concerning all the elections that have the selected configuration. In the *Configuration Selection* menu, cf. Figure A.11, it is possible to select the election configuration and if it is necessary to create the elections' keys and/or the *Administrators* signing keys. If the keys are not in the *Commissioner* database an error message will appear. For security reasons the administrators signing keys should be created by the *Administrators* and not by the *Commissioner*, cf. Section A.5.1.

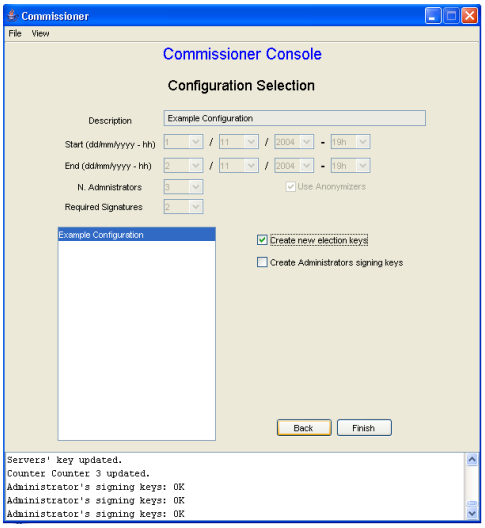

Figure A.11: Configuration Selection (create databases)

After selecting the configuration press the *Finish* button to create the data files. The following files will be created:

1. One encrypted file containing the elections' private keys.

2. One file containing the decryption key to decipher the elections' private keys file.

3. One file for each enabled *Administrator*.

4. One file for the *Ballot Distributors*.

5. One file for the *Anonymizers* and *Counters*.

6. One file containing a list of the active *Ballot Distributors*.

7. One file containing a list of the active *Counters*.

All files are signed by the *Commissioner*.

The next step is to setup the servers' databases whith the created files. To load the files into a server's database, launch the server, cf. Section A.3.2, select the *U - Update database* option and enter the file name. Now the server is ready to be started, just select the option *S - Start server*.

An additional step is required to start the *Anonymizer* server, it is necessary to select the *Counter* to which forward the votes. Select the option *F - Forward Counter selection* and enter the name of the file containing the list of active *Counters*, then select one. Now the server is ready to be started, just select the option *S - Start server*. It will be asked for the maximum number of ballots to be sent after each delay and the maximum delay time, cf. Section 5.3.1.

A.5.1 Administrators signing keys

If the signing keys are created by the *Commissioner* is it possible for the *Commissioner* to keep the signing keys and use them produce valid votes, corrupting the election by it self. Therefore, is is recommended the creation

of the signing keys by the *Administrators* and then import the verification keys to the *Commissioner*. The steps needed are the following:

1. Export the elections list to a file. Go to the *Utilities* menu and select the *Export Elections* option.

2. Create the signing keys for each *Administrator*. Start the *Administrator* server (cf. Section A.3.2) and select the *K - Create signing keys* option. Then use the file saved in step one as input. The output is a file containing the signature verification keys.

3. Import the signature verification keys. In the *Utilities* menu and select the *Import Administrators Signing Keys* option.

A.6 Voting process

A.6.1 Start the *Voter's Module*

To install the *Voter's Module* copy the *revs_voter.jar* file to the installation directory and the following files to the *"conf"* subdirectory in the installation directory:

- *distributors*: the file containing the active Ballot Distributors list (cf. Section A.5).

- *policy.txt*: this file is a Java policy file, for more information about it consult the Java documentation. An example of a policy file is available at REVS download site.

- *commissioner.crt*: this file contains the commissioner public key certificate signed by the CA.

- *tstore.ks*: this file contains the CA public key certificate. To create this file follow the instructions in cf. Section A.2.3.

- *welcome.html*: this file contains the welcome message, formatted in HTML that appears on the welcome screen of the *Voter's Module* (Figure A.12).

A.6.2 Voting steps

The voting steps are the following:

1. Start the *Voter's Module* with the following command:
   java -classpath "voter.jar" -Djava.security.policy=conf/policy.txt inescID.revs.voter.VoterEngine
   A welcome screen should appear (Figure A.12). To continue press *OK*.

2. Then the voter authentication is requested (Figure A.13). To continue press *OK*. A voter authentication confirmation should appear (Figure A.14), to confirm press *Yes*.

3. The next screen presents the list of elections in which the voter can participate. The voter should pick one an press *OK* to continue (Figure A.15).

4. Now it is displayed the ballot (Figure A.16). The voter should fill in the ballot and when done press *OK* to submit the vote.

5. A validate confirmation message will appear (Figure A.19). After the confirmation the vote is send to the Administrators for signing, but before that it is possible to save the voting state, cf. Figure A.18, which is necessary to recover the voting process in the case of being impossible to submit the vote.

6. After collecting the administrators signatures it will appear a submit confirmation message (Figure A.19). The vote is only submitted after this confirmation. If the voter does not confirm the submission, the submission is aborted. To resume the submit process it will be necessary the previously saved voting state.

7. Finally it is displayed the voting process report (Figure A.20). From this menu it is possible to go to the election selection menu or to the welcome message menu.

If the vote cannot be submitted successfully there will be an error message on voting process report. To resume the voting protocol go to the *File* menu, in the welcome screen, and select the *Resume Voting* option (Figure A.21). Then the authentication menu should appear and the voting process is resumed.

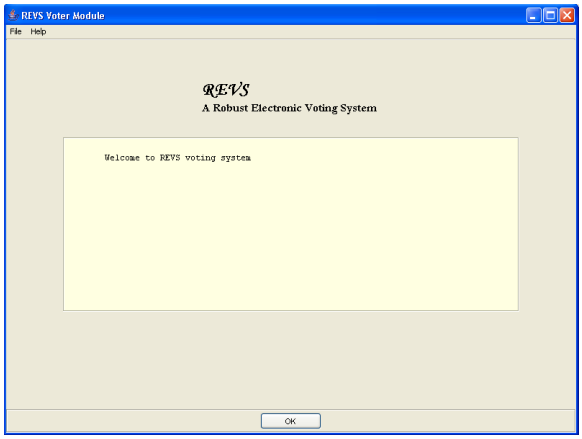

Figure A.12: Welcome screen

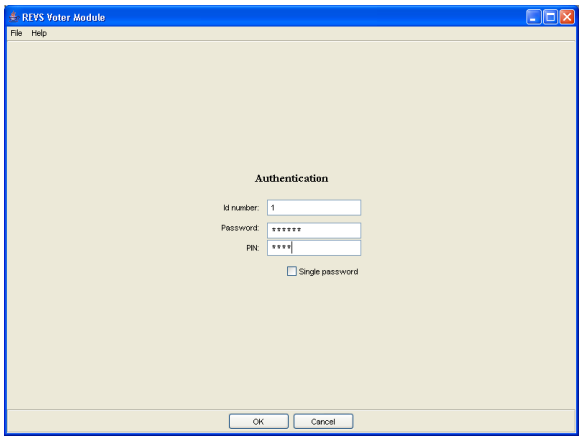

Figure A.13: Authentication screen

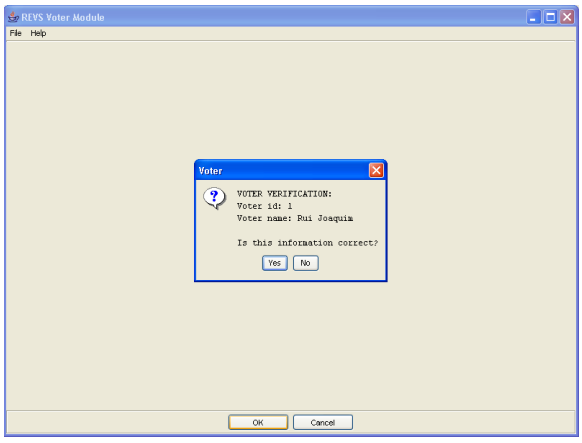

Figure A.14: Authentication confirmation

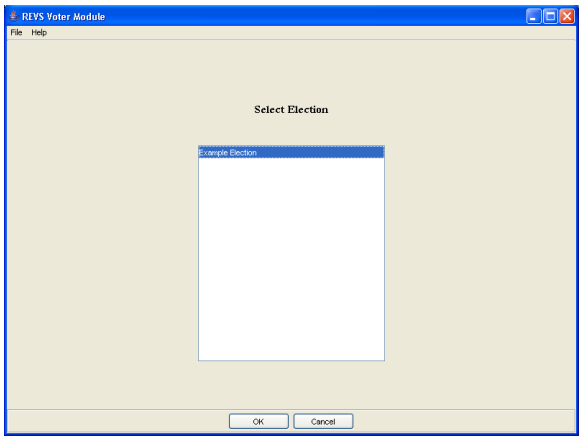

Figure A.15: Election selection screen

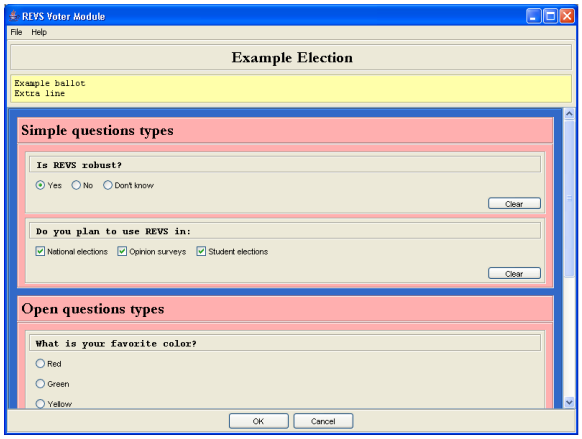

Figure A.16: Ballot display

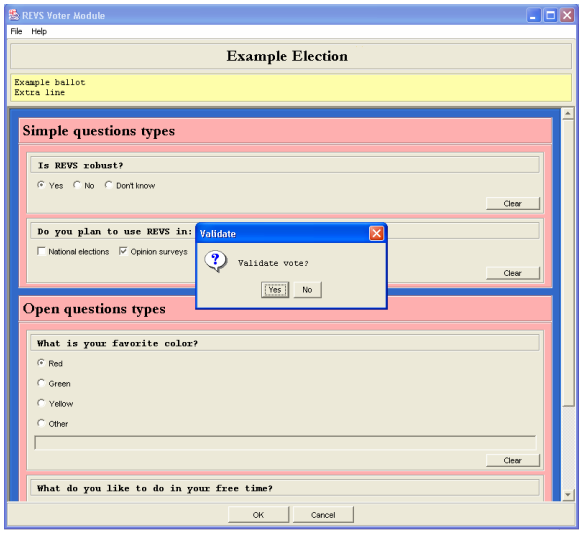

Figure A.17: Validate confirmation

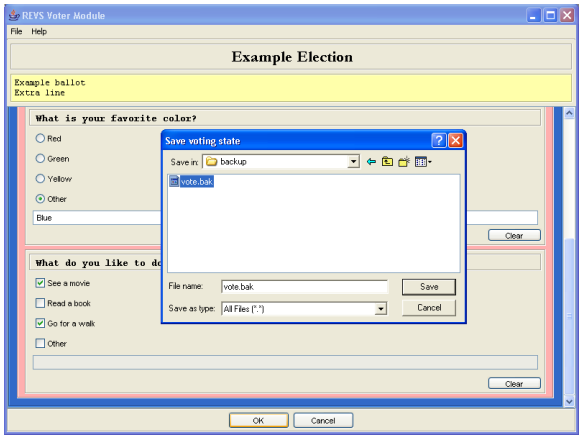

Figure A.18: Save vote state

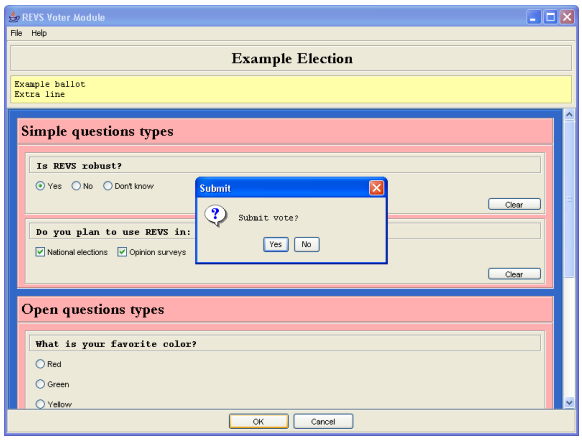

Figure A.19: Submit confirmation

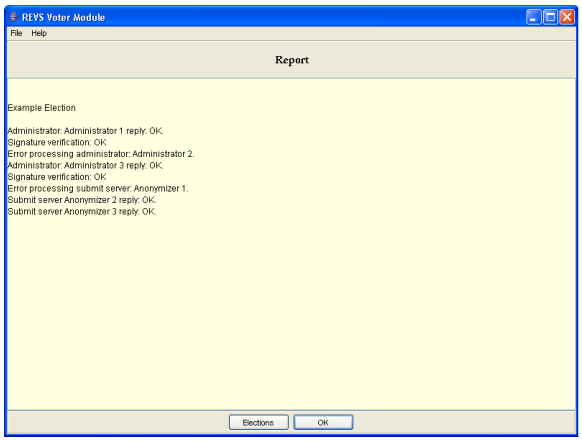

Figure A.20: Report screen

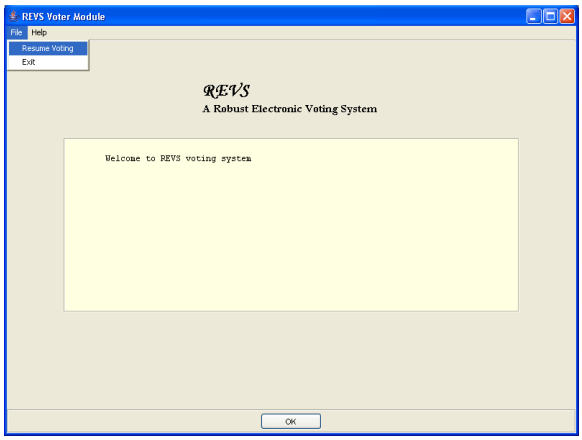

Figure A.21: Resume voting

A.7 Election tally

After the election polling close select the *Counter's* option *T - Tally votes* to decipher the votes, verify the *Administrators'* signatures and to produce the final election tally. For this action operation it will be needed the file containing the encrypted elections' private keys and the file containing the decryption key for the first one.

To view the results open the file *index.htm* in the *"results"* directory, a resume table of the elections results will appear (Figure A.22). There it is possible to choose two views of the elections results (Figures A.23 and A.24).

The *Counters'* option *G - Gather votes* should be used if there were multiple counters used in election to gather the voter from all of them. For this task it is necessary the file containing the list of active counters.

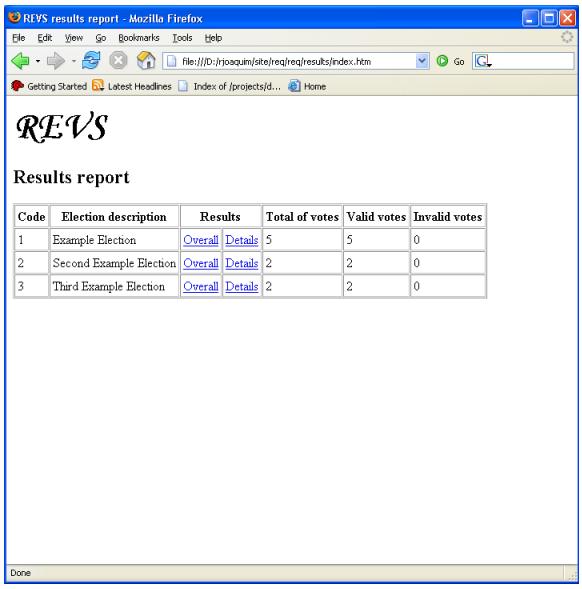

Figure A.22: Results resume table

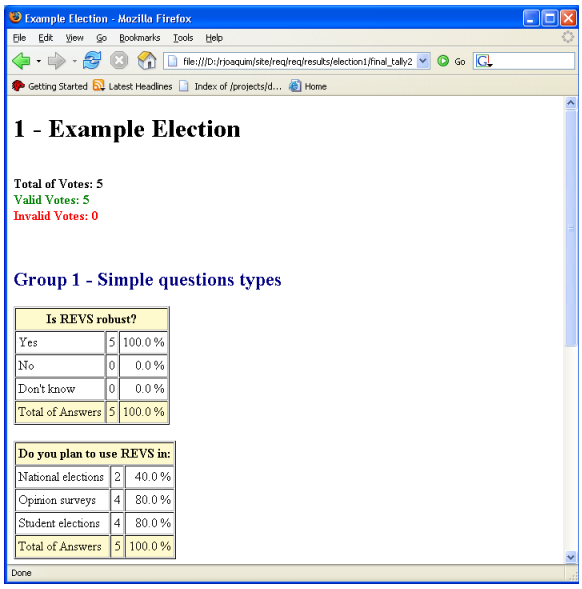

Figure A.23: Overall results

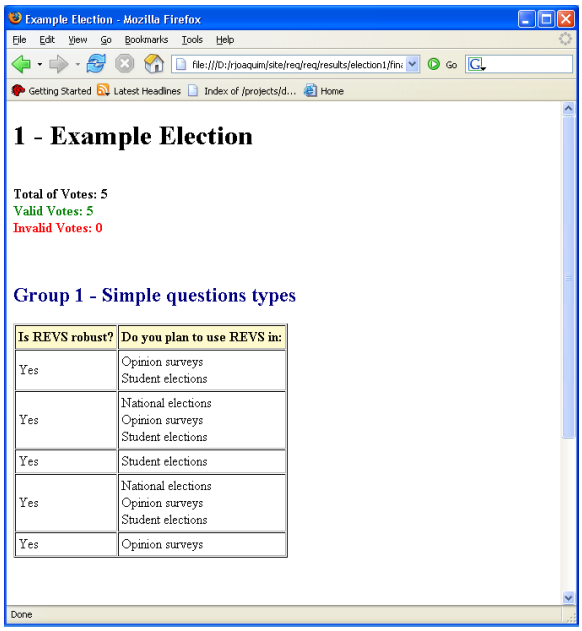

Figure A.24: Results details

Bibliography

[Abe98] M Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In *Advances in Cryptology – Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 437–447, Helsinki, Finland, 31 May– 4 June 1998. Springer-Verlag.

[Abe99] Masayuki Abe. Mix-networks on permutation networks. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *Advances in Cryptology - ASIACRYPT '99, International Conference on the Theory and Applications of Cryptology and Information Security*, volume 1716 of *Lecture Notes in Computer Science*. Springer, 1999.

[AH01] Masayuki Abe and Fumitaka Hoshino. Remarks on mix-network based on permutation networks. In *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography*, volume 1992 of *LNCS*, pages 317–?? Springer, 2001.

[BCR87] G. Brassard, C. Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In A.M. Odlyzko, editor, *Proceedings of*

Advances in Cryptology – CRYPTO '86, volume 263 of *LNCS*, pages 234–238. Springer-Verlag, 1987.

[BF85] J. Benaloh and M. Fischer. A robust and verifiable cryptographically secure election scheme. In *Proceedings of 26th IEEE Symposium on Foundations of Computer Science - FOCS '85*, pages 372–382. IEEE Computer Society, 1985.

[BFP+01] Baudron, Fouque, Pointcheval, Stern, and Poupard. Practical multi-candidate election system. In *PODC: 20th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2001.

[BG02] Dan Boneh and Philippe Golle. Almost entirely correct mixing with applications to voting. In Vijay Atlury, editor, *Proceedings of the 9th ACM Conference on Computer and Communication Security (CCS-02)*, pages 68–77, New York, November 18–22 2002. ACM Press.

[BO2] Bo2k. http://www.bo2k.com.

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 544–553, Montréal, Québec, Canada, 23–25 May 1994.

[BY86] J. Benaloh and M. Yung. Distributing the power of government to enhance the power of voters. In *Proc. 5th ACM Symp. on Principles of Distributed Computation*, pages 52–62. ACM, 1986.

[Cal00] California Internet Voting Task Force. A report on the feasibility of internet voting. http://www.ss.ca.gov/executive/ivote, 18 January 2000.

[Cal01] Caltech-MIT Voting Technology Project. Voting - What Is, What Could Be. http://www.vote.caltech.edu/Reports, July 2001.

[CC97] L. Cranor and R. Cytron. Sensus: A security-conscious electronic polling system for the internet. In *Proceedings of the Hawaii International Conference on System Sciences. Wailea, Hawaii.*, 1997.

[CFSY96] Ronald Cramer, Matthew Franklin, Berry Schoenmakes, and Moti Yung. Multi-authority secret-ballot elections with linear work. In Ueli Maurer, editor, *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 12–16 May 1996.

[CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology – EUROCRYPT '97*, pages 103–118, 1997.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[Cha82] David Chaum. Blind signatures for untraceable payments. In Rivest R. L. Chaum, D. and A. T. Sherman, editors, *Proceedings of Advances in Cryptology – CRYPTO '82 (New York)*, pages 199–203. Plenum Press, 1982.

[Cha88] David Chaum. Elections with unconditionally- secret ballots and disruption equivalent to breaking rsa. In *Proceedings of Advances in Cryptology – Eurocrypt'88*, volume 300 of *LNCS*, pages 177–182. Springer-Verlag, 1988.

[Coh87] Josh D. Cohen. *Verifiable secret-ballot elections /–Josh Daniel Cohen Benaloh.* PhD thesis, Yale University, 1987, 1987.

[Cra96] Lorrie Faith Cranor. Electronic voting — computerized polls may save money, protect privacy. *ACM Crossroads*, 2(4), April 1996.

[Cra01] Lorrie F. Cranor. Voting after florida: No easy answers. *Ubiquity: An ACM IT Magazine and Forum*, (47), February 2001.

[DJ01] Ivan B. Damgård and Mads J. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. pages 119–136, 2001. RS-00-45.

[DuR99] Brandon DuRette. Multiple administrators for electronic voting. Bachelor's thesis, Massachusetts Institute of Technology, May 1999.

[ElG85] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Trans. on Information Theory*, 31(4):469–472, July 1985.

[FOO92] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In Jennifer Seberry and Yuliang Zheng, editors, *Advances in Cryptology—AUSCRYPT 92*, volume 718 of *LNCS*, pages 244–251. Springer-Verlag, 13–16 December 1992.

[FS01] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO ' 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2001.

[GMR85] S. Goldwasser, S Micali, and C. Rackoff. The knowledge complexity of interactive proofs. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 291–305, 6–8 May 1985.

[Gol95] Oded Goldreich. Foundations of Cryptography - Fragments of a Book. http://www.wisdom.weizmann.ac.il/ oded/homepage.html, 1995.

[Her97] Mark Herschberg. Secure electronic voting over the world wide web. Master's thesis, Massachusetts Institute of Technology, May 1997.

[HS98] Qi He and Zhongmin Su. A new practical secure e-voting scheme. In *Proceedings of 14th International Information Security Conference (SEC'98)*, 1998.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in cryptology—EUROCRYPT 2000 (Bruges)*, volume 1807 of *Lecture Notes in Comput. Sci.*, pages 539–556. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2000.

[Int01] Internet Policy Institute. Report of the National Workshop on Internet Voting: Issues and Research Agenda. http://www.diggov.org/archive/library/dgo2000/dir/PDF/vote.pdf, March 2001.

[Jak98] M. Jakobsson. A practical mix. In *Advances in Cryptology – EUROCRYPT '98*, pages 448–461, 1998.

[Jak99] Markus Jakobsson. Flash mixing. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '99)*, pages 83–90, New York, May 1999. Association for Computing Machinery.

[JJ01] Jakobsson and Juels. An optimally robust hybrid mix network. In *PODC: 20th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2001.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium (SECURITY-02)*, pages 339–353, Berkeley, CA, USA, August 5–9 2002. USENIX Association.

[KKP03] Robert Kofler, Robert Krimmer, and Alexamder Prosser. Electronic voting: Algorithmic and implementation issues. In *36th Hawaii International Conference on System Sciences*, 2003.

[LJZF04] Ricardo Lebre, Rui Joaquim, André Zúquete, and Paulo Ferreira. Internet voting: Improving resistance to malicious servers in REVS. In *IADIS International Conference on Applied Computing*, March 2004.

[LK02] Lee and Kim. Receipt-free electronic voting scheme with a tamper-resistant randomizer. In *ICISC: International Conference on Information Security and Cryptology*. LNCS, 2002.

[LMMM00] A. Lioy, F. Maino, M. Marian, and D. Mazzocchi. DNS security. In *TERENA Networking Conference*, pages 22–25, 2000.

[MK00] Mitomo and Kurosawa. Attack for flash MIX. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT '00: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 2000.

[MOV97] A. J. Menezes, P. C. Van Oorschot, and A. S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[MSOA01] A. Monteiro, N. Soares, R. Oliveira, and P. Antunes. Sistemas electrónicos de votação. FCT/UL Technical Report DI-FCUL-TR-01-9, October 2001.

[Mür00] O. Mürk. Electronic Voting Schemes. Semester work. http://www.cs.ut.ee/ olegm/, 2000.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Pierangela Samarati, editor, *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 116–125, Philadelphia, PA, USA, November 2001. ACM Press.

[NSS91] H. Nurmi, A. Salomaa, and L. Santean. Secret ballot elections in computer networks. *Computers and Security*, 36(10):553–560, 1991.

[Oka96] Tatsuaki Okamoto. An electronic voting scheme. In *Proceedings of IFIP '96*, pages 21–30. Chapman & Hall, 1996.

[Oka97] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Proceedings of 5th International Computer Security Conference*, pages 25–35, 1997.

[OKST97] W Ogata, K Kurosawa, K Sako, and K Takatani. Fault tolerant anonymous channel. In *Information and Communications Security — First International Conference*, volume 1334 of *Lecture Notes in Computer Science*, pages 440–444, Beijing, China, 11–14 November 1997. Springer-Verlag.

[OMA+99] Ohkubo, Miura, Abe, Fujioka, and Okamoto. An improvement on a practical secret voting scheme. In *ISW: International Workshop on Information Security*, LNCS. Springer-Verlag, 1999.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Lecture Notes in Computer Science*, 1592:223–238, 1999.

[PIK93] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In Tor Helleseth, editor, *Advances in Cryptology—EUROCRYPT 93*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer-Verlag, 1994, 23–27 May 1993.

[Riv01] R. Rivest. Electronic Voting. In *Proceedings of Financial Cryptography'01. Grand Cayman, Cayman Islands*, 2001.

[RSA77] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. MIT LCS Technical Report MIT/LCS/TM-82, 1977.

[Rub02] A. Rubin. Security Considerations for Remote Electronic Voting Over the Internet. *Communications of the ACM*, 45(12), 2002.

[Sal91] A. Salomaa. Verifying and recasting secret ballots in computer networks. In Hermann Maurer, editor, *Proceedings of New Results and New Trends in Computer Science*, volume 555 of *LNCS*, pages 283–289, Berlin, Germany, June 1991. Springer.

[SK94] Kazue Sako and Joe Kilian. Secure voting using partially compatible homomorphisms. In Yvo G. Desmedt, editor, *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 411–424. Springer-Verlag, 21–25 August 1994.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme: A practical solution to the implementation of a voting booth. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology—EUROCRYPT 95*, volume 921 of *LNCS*, pages 393–403. Springer-Verlag, 21–25 May 1995.

[UKD] Uk-edemocracy. http://www.edemocracy.gov.uk.