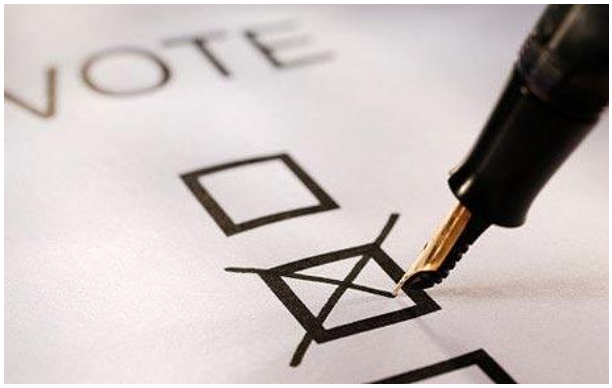
TÉCNICO LISBOA



Scroll, Match & Vote:
An E2E Coercion Resistant Mobile Voting System

Gonçalo David Martins Tourais Pereira

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisor: Prof. Carlos Nuno da Cruz Ribeiro

Examination Committee

Chairperson: Prof. Paulo Jorge Pires Ferreira
Supervisor: Prof. Carlos Nuno da Cruz Ribeiro
Member of the Committee: Prof. Manuel Eduardo Correia

June 2014

Acknowledgments

Em primeiro lugar gostaria de agradecer ao meu orientador Professor Carlos Ribeiro que me deu um apoio incondicional em todas as fases da tese. Também gostaria de agradecer a ajuda e disponibilidade que o Rui Joaquim, André Brioso e Nuno Pinheiro me deram no esclarecimento das dúvidas que foram surgindo e que prontamente foram respondidas.

Quero obviamente prestar o maior agradecimento ao meu pai, Gabriel Pereira, e à minha mãe a "santa da programação", Etelvina Pereira, por todo o apoio, força, ajuda e amor que me deram todos os dias da minha vida. Sem eles nada seria possível. Uma especial mensão ao meu irmão, Pedro Pereira, por todas as conversas de teor aleatório e planos para termos uma vida melhor e mais feliz.

Agradeço a todos os meus colegas que me animaram e conseguiram dar força para atinguir os meus objectivos. Queria fazer uma mensão especial à Nádia Gonçalves pelas batalhas infinitas que temos enfrentado e por me ter tornado uma melhor pessoa, ao Luís Pedro e à Joana Alves e pelo companheirismo, amizade e felicidade sempre presente durante todos estes anos e uma palavra de amizade ao Vitor Mansur, Roberto Silva, João Rosa, André Camões, Luís Luciano, João Salada, Carlos Simões, Marco Alves, Bruno Henriques e João Amaro que por todos estes anos me acompanharam nesta grande aventura que foi o IST.

Abstract

Mobile Internet Elections are appealing for several reasons: they promise voter convenience, lower abstention rates and lower costs. However, there are a number of trust issues that prevent them from becoming ubiquitous, the most relevant of which is the possibility of coercion of the voter at the time of the vote. But other issues, like the trustworthiness of both the services running the election and the mobile voting platform (usually the voter's computer or smartphone), are also major barriers to their adoption.

The proposed "Scroll, Match & Vote" (SM&V) system aims to overcome these trust issues, while ensuring the usability needed for wide adoption. SM&V builds upon previous e-voting solutions that ensure end-to-end verifiability and collusion resistance [1], and adds coercion resistance in a degree similar to the one of traditional booth voting systems.

SM&V requires the use of a device with Internet connection and a multitouch screen e.g. a smartphone. In the voting phase the voter is shown two lists side by side on the device. One of the lists contains all the election candidates and the other vote codes. One of these vote codes is correct the others are false. The voter votes by scrolling one or both lists and match her chosen candidate with the correct code.

Keywords: Mobile Voting System, End-to-End Verifiability, Coercion Resistance

Resumo

Os sistemas de votação eletrónicos móveis com Internet são apelativos por várias razões: prometem maior comodidade ao eleitor, promovem menores níveis de abstenção e redução de custos. No entanto, há uma série de problemas a nível de confiança que os impedem de se tornarem omnipresentes, o mais relevante dos quais é a possibilidade de coação do eleitor no momento da votação. Mas outras questões, como a confiabilidade de ambos os serviços que suportam a eleição e a plataforma de votação móvel (geralmente o computador do eleitor ou *smartphone*), são também grandes barreiras à sua adoção.

O sistema proposto "Scroll, Match & Vote" (SM&V) tem como objectivo ultrapassar estes problemas de confiança, enquanto garante a usabilidade necessária para uma vasta adoção. SM&V é construído sobre uma solução prévia de votação eletrónica que garante verificabilidade ponta a ponta e resistência a conluio [1] e adiciona resistência a coação a um nível equiparável ao dos sistemas de votação tradicionais.

SM&V requer o uso de um dispositivo com ligação à Internet e um ecrã multitáctil e.g. um *smartphone*. Na fase de votação, é apresentado ao votante duas listas juntas no dispositivo. Uma das listas é composta por todos os possíveis candidatos e a outra por códigos de votação. Um destes códigos é correcto, os outros são falsos. O votante vota rodando uma ou ambas as listas fazendo corresponder o candidato pretendido com o código correcto.

Palavras-chave: Sistema de Votação Móvel, Verificabilidade Ponta a Ponta, Resistência a Coação

Contents

List of Figures

List of Tables

List of Acronyms

ADT Android Development Tools

AID Application Identifier

APDU Application Packet Data Unit

API Application Programming Interface

BB Bulletin Board

BBox Ballot Box

BLOB Binary Large Object

CC Code Card

DTK Development Tool Kit

EC Electoral Commission

ECApp Electoral Commission Application

EEPROM Electrically Erasable Programmable Read-Only Memory

ER Election Register

HO Helper Organization

IDE Interface Development Kit

IO Independent Organizations

JPA Java Persistence API

JSF Java Server Faces

MP3 Mark Pledge 3

NFC Near Field Communication

OTA Over-the-Air

PDD Pledge Display Device

RAM Random Access Memory

SDCard Secure Digital Card

SDK Software Development Kit

SE Secure Element

SOAP Simple Object Access Protocol

UICC Universal Integrated Circuit Card

VST Voter Security Token

XSD XML Schema Definition

WSDL Web Service Definition Language

Chapter 1

Introduction

Democracy is a form of government in which all electors have an equal say in the decisions that affect their lives. In our world most of the countries follow this system of government. The most common way of voting in a democratic country is made in controlled voting precincts, that must be made in a specific day.

One of the main problems of current western hemisphere general elections is the high abstention rate [2]. I argue that the space and time constraints of most elections increases that problem, and that by introducing internet voting these constraints may be mitigated.

Internet elections have been a research subject for many years leading to many interesting results. In spite of the risks to voter's privacy and election integrity, evidence seems to point out that Internet voting has come to stay. Accordingly to 2007 study [3], numerous Internet elections ($\sim$ 139) had already occurred worldwide and many of them ($\sim$ 40%) were actual real binding elections. These numbers have been increasing as more countries tried or adopted the Internet voting channel. A notable example is Estonia's case which is moving to/already have national binding Internet elections. A more recent example is Norway which ran an Internet voting system in 2011 [4].

The arguments in favor of Internet elections are obvious: i) increased voter convenience and participation, ii) tally accuracy and speed, iii) reduced costs, among others. However, the arguments against Internet elections are pertinent : i) the insecure voting platform problem, which results from the use of multipurpose devices owned and managed by the voter [5]; ii) the lack of transparency resulting from the nonexistence of physical votes and the possibility of collusion between the digital devices participating in the election; and iii) the nonexistence of private voting precincts that paves the way for several coercion scenarios.

The widespread use of smartphones with ubiquitous Internet access stressed, some of these advantages and disadvantages. While it is even more convenient for the voter to vote on her own smartphone, it is also easier for the coercer, given that the voter may vote anywhere. In spite of this, one of the most common reasons for failure of voting experiments is the lack of usability; voting systems that are too complex are deemed to fail, even if they are able to overcome all the above security disadvantages [6].

Scroll, Match & Vote (SM&V) is an end-to-end verifiable [7, 8] and collusion-resistant Internet voting

system [1], with coercion resistant properties, although with a reduced mobility when compared with other Internet voting systems [1], given that part of the process must be performed in a controlled precinct.

Elections are usually constrained in time and space, i.e. they must be carried at the election-day and in controlled precincts. This double constraint is a source of abstention, given that not every voter is available to be at a specific place at that specific time. Removing either of these constraints is very problematic. If the election takes too long (i.e. several months) democracy gets injured because some voters vote with much less information than others. Early and postal-voting are seen as exceptions, not as rule. Removing the space-constraint is also difficult because it usually means loosing coercion resistance [9]. The current proposal follows the path of JCJ/Civitas [10, 9] and splits the two constraints such that the space-constraint and the time-constraint do not apply to the same action. The voter must register at a private booth without tight time constraints (within the timespan of one or two months) and must vote at election-day without any space constraints (with the exception of having Internet connection).

SM&V assumes that the voter owns a mobile Internet device with a multitouch screen (from now on referred as the voter's smartphone) with a secure element (either the UICC, an SDCard or an embed secure element) and proposes a new voting interface with coercion resistant properties that in combination with a variant of the EVIV voting protocol [1] attains also end-to-end verifiability and some resilience to collusion.

In the voting phase two lists of codes are shown to the voter side by side on the smartphone. One of the lists contains all the candidates and the other vote codes. One of these vote codes is correct (dubbed "pledge") the others are false. The voter votes by scrolling one or both lists and match her chosen candidate with the "pledge".

SM&V ensures integrity of the vote provided that t out of n configurable trustees are trustworthy. However, if the voter is coerced the protocol ensures confidentiality and integrity only if the secure element is trustworthy, thus in coercion scenarios the protocol is not collusion-resistant given that it only takes the secure element and the coercer to break integrity. Notice that this is a fundamental problem of end-to-end verifiable protocols; they ensure integrity only if the voter is able to verify the vote, if the voter is coerced not to verify they do not ensure integrity.

With SM&V, a voter may register several times and vote also several times, only the last one of both actions counts. The voter may also choose to register with SM&V and then choose another method to vote.

The SM&V prototype is based on the EVIV implementation. The applet for the secure element was created based on the EVIV applet, which had to be improved, to mitigate some memory problems encountered, and all the SM&V logic was implemented. Furthermore, an Android application was created from scratch with the correspondent interactions with the servers and secure element. Finally, several bugs in the EVIV server architecture were solved other unimplemented functions were implemented in order to make an fully integrated voting system.

The next chapter presents the most relevant techniques and protocols used in the construction of cryptographic voting systems for an easier understanding of the next chapters. Chapter 3 discusses

some recent internet voting proposals and their contribution for this dissertation. Section 4 describes the SM&V protocol architecture, their elements, its properties as well as its trust model. In chapter 5 and 6 the SM&V voting protocol is described and the implementation is fully detailed while chapter 7 presents the evaluations done to validate this dissertation. Finally the document is concluded in chapter 8. There is an appendix for the proofs of the properties identified in section 4.2.

Chapter 2

General Concepts

This chapter presents the most relevant techniques and protocols used in the construction of cryptographic voting systems. The main purpose is to provide the reader the necessary basis for an easier understanding of the following chapters.

2.1 Homomorphic Cryptosystem

The main idea of the homomorphic encryption is the application of algebraic operations without deciphering previously [11]. The final result can the deciphered with the correspondent secret key of the entity without anyone knowing what were the initial numbers of the operation while the same secret key cannot decipher the initial numbers.

In this thesis the main usage of the homomorphism is made in the exponential ElGamal:

Additive homomorphism between two encryptions:

$$\mathcal{E}_{pk}(m_1, r_1) \oplus \mathcal{E}_{pk}(m_2, r_2) = \mathcal{E}_{pk}((m_1 + m_2)\ mod\ q, (r_1 + r_2)\ mod\ q)$$

Multiplicative homomorphism between an encryption and a value n:

$$\mathcal{E}_{pk}(m, r) \otimes n = \mathcal{E}_{pk}((m \cdot n)\ mod\ q, (r \cdot n)\ mod\ q)$$

In the above expressions the $mod\ p$ notation was omitted for simplicity.

The Homomorphic Cryptosystem is an important concept since the SM&V uses it to count the votes without breaking the privacy. So the system only counts the encrypted votes and a final decipher is only needed to get the final election result.

2.2 Exponential ElGamal

The ElGamal scheme was introduced in 1985 by Taher ElGamal [12] and it is an asymmetric key encryption algorithm for public key cryptography which is is based on the Diffie-Hellman key exchange. In ElGamal, security is gained in the complexity of the mathematical algorithm.

What motivated the creation of the Exponential ElGamal was the possibility to offer the characteristics of ElGamal, but adding the possibility to perform sums with the chipertext without deciphering (homomorphic addition) and homomorphic multiplication with a constant [13] (Section 2.1).

The public parameters of the system are the q and p, both big prime numbers. The ElGamal key pair consists of a private key sk and the corresponding public key $pk = h = g^{sk}\ mod\ p$. The private key sk is a randomly chosen integer such that $0 < sk < q$.

In Exponential ElGamal, homomorphic algebraic operations can be made encrypting g^m instead of m for some generator g (usually the same one used to generate the public key).

Exponential ElGamal encryption algorithm $\mathcal{E}$:

$$c = \langle x, y \rangle = \langle g^r\ mod\ p, g^m \cdot h^r\ mod\ p \rangle = \mathcal{E}_{pk}(m, r), r \xleftarrow{R} \mathbb{Z}_q$$

Exponential ElGamal decryption algorithm $\mathcal{D}$:

$$m = log_g(g^m\ mod\ p) = log_g(\frac{y}{x^{sk}}\ mod\ p) = \mathcal{D}_{sk}(c)$$

Running the standard decryption gives g^m and recovering m requires to solve the discrete log. As long as m is small, this can be done algorithmically.

Paillier has homomorphic properties as well, and can support a proper decryption of any sized message. Despite this, many voting schemes still use exponential ElGamal because it is faster, easier to do distributed key generation, and not patented. Exponential ElGamal algorithm works in the Universal Integrated Circuit Card (UICC) Card processor that is necessary for SM&V protocol.

2.3 Mix-Nets

The Mix-Nets concept was introduced by David Chaum in 1981 [14] with the main objective to guarantee the privacy of the information, decoupling the message from the sender. This solution assumes that at least one of the different entities, used by the mix-net, is honest.

Mix-Nets are commonly used to provide an anonymous communication channel between the voters and the election authorities or to anonymize the encrypted votes before they are decrypted for the tally.

Each of these entities is an anonymity server called the Mix-Server or Mixes, and its main task is to keep, transform, delay, mix and resent a sequence of received ciphered messages.

The processes start when the message is encrypted and is ciphered several times with the public keys of the mixes. The chipertext message will be decrypt in the reverse order of the mix re-encryptions,

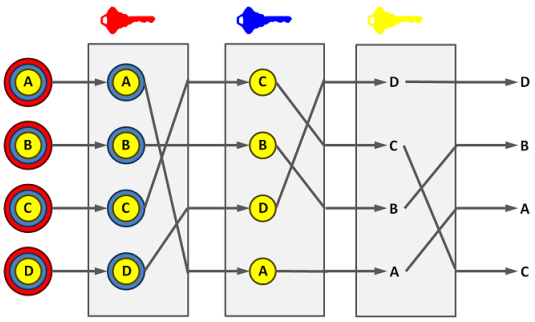


Figure 2.1: Mix-Net Decryption

so the path of the message is normally previously defined. When the message reaches the entities, it is mixed in a way that is impossible to associate the input information with the output information as it can be seen in Figure 2.1.

Since it is not possible to verify the mix process of the entity, mix-nets must provide a zero-knowledge proof [15] of correct shuffling, i.e. prove that the mix-net output messages are re-encryptions of the mix-net input without revealing the shuffle performance.

One of the main disadvantages of this technique is the large computational process.

2.4 Blind Signatures

Blind signatures is a type of digital signature proposed by Chaum in 1982 [16]. The main focus of this technique is to provide, as the name implies, a signature on the content of a message without the signer comes to know its contents. The blind signature is commonly used to validate the vote without revealing it.

Making the analogy to paper-based world, getting a blind signature is similar to setting a document bellow a sheet of carbon inside an envelope, if someone signs the outside the envelope, it also signs the document inside the envelope.

In a more formal way e.g. Alice, the voter, wants the Election Authority to sign the message for her to send afterwards to the Voting Center:

1. Alice chooses a blind factor r between 1 and n;

2. Alice send the message's digest cyphered so that the Election Authority do not know who she voted to

$$B = Hash(m) \cdot r^{pk}$$

3. After the Election Authority receives the message's digest, it checks for the credentials for the authentication process and curb duplicates;

4. Election Authority signs the ciphered message's digest and sends back to Alice

5. Alice deciphers the message's digest and obtains the sign message's digest by the Election Authority

$$Sign(B) = (Hash(m) \cdot r^{pk})^{sk} \Leftrightarrow Sign(B) = Hash(m)^{sk} \cdot r \Leftrightarrow Sign(m) = Sign(B)/r$$

6. Alice sends the message and the sign message's digest to a mix relay to decouple the identity of the sender from the message to the Voting Center to preserve the anonymity

2.5 Threshold Public key Cryptosystem

The threshold public key cryptosystem is a technique to create a private key shared among a set of n "trustees". To decrypt the message it is needed the collaboration of a subset of t trustees. None of the entities is able to reconstruct by itself the secret key sk.

To create the shared secret key it is used a secret sharing concept introduced by Shamir in 1979 [17]. Shared secret key generation and message decryption protocols, for the ElGamal cryptosystem, were first proposed by Pedersen in [18]. Those algorithms were used in the exponential ElGamal variant of [19]. Follows a brief description of the key generation and message decryption protocols:

Key Generation: In the key generation protocol each trustee T_i computes a secret share $s_i \in \mathbb{Z}_q$ and commits to it by time the $h_i = g^{si}$ are published. Additionally, each trustee secretly shares its secret among the other $n-1$ trustees such that the secret key sk can be reconstructed from any set Λ if t trustees using the appropriate Lagrange coefficients:

$$sk = \sum j \in \Lambda s_j \cdot \lambda_{j,Lambda}$$
$$\lambda_{j,\Lambda} = \prod_{l \in \Lambda / \{j\}} \frac{l}{l-j}$$

The resulting election public key is $pk = k = \sum h_i$

Message decryption: To decrypt a message $\mathcal{E}_{pk}(m,r) = (x,y)$ without reconstructing the secret key sk, the trustees do the following:

1. Each trustee T_i publishes $w_i = x^{si}$ and proves in zero knowledge that:

$$log_g\ h_i = log_x\ w_i$$

2. Let Λ denote any subset of t trustees who showed a valid the zero-knowledge proof. The message m can be recovered as:

$$m = y / \prod_{i \in \Lambda} w_i^{\lambda_{j,\Lambda}}$$

2.6 Bulletin Board

The bulletin board is a component usually used in e-voting. The main purpose is to have transparency in the electoral process. All the information can be checked publicly by anyone and all the published data is authenticated.

Additionally the bulletin board must assure availability by replication and guarantee that the data in there cannot be deleted. Since it is only possible to add information, anyone can check the evolution of the data.

2.7 MarkPledge 3

MarkPledge was presented by Andrew Neff in 2004 [7] with the goal of providing high vote encryption assurance to the voter. MarkPledge is currently in version 3 [20]. It is the only version that may run on a voting machine with reduce computation and memory resources [7, 21, 20].

In Mark Pledge 3 (MP3) the ballot creation device is not trustworthy, it must prove to the voter that the ballot is well constructed and that the vote is cast-as-intend. This proof is done in two steps. The first step is the actual creation of the ballot together with the proofs (of well-formed ballot). The second step happens after the ballot is created, and proves that the vote is cast-as-intend. The next paragraphs describe these two steps and the tally process in detail.

Ballot Creation: An MP3 ballot is the concatenation of k candidate votes $cv_i, i = 1, \ldots, k$ together with a proof $\mathcal{P}1$ that only one of the cv_i is a $YESvote$ and all the others are $NOvotes$ (for details on this proof check [20]).

Each candidate vote $cv_i = \langle cvote_i, \mathcal{P}2_i \rangle$ is comprised by a candidate vote encryption $cvote_i$ and a proof ($\mathcal{P}2_i$) that the candidate vote encryption is either a $YESvote$ or a $NOvote$ ($\mathcal{P}1$ assumes that $\forall_i \, \mathcal{P}2_i$ apply)(for details on $\mathcal{P}2_i$ check [20, 19]).

An MP3 candidate vote encryption $cvote_i = \langle u_i, v_i \rangle$ is composed by two independent encryptions: $u_i = \mathcal{E}_{pk}(b_i, \tau)$ is the encryption of either $b_i = 1$ for a $YESvote$ or $b_i = -1$ for a $NOvote$; $v_i = \mathcal{E}_{pk}(\theta_i, \delta)$ is the encryption of a random confirmation (commit) code ($\theta_i \in_R \mathbb{Z}_q$), which in the case of a $YESvote$ is pledged to the voter. Both encryptions use exponential ElGamal with the public key of the election pk and randomization factors τ and δ, respectively.

$$Ballot = cv_0, \ldots, cv_k, \mathcal{P}1$$
$$cv_i = \mathcal{E}_{pk}(b_i, \tau), \mathcal{E}_{pk}(\theta_i, \delta), \mathcal{P}2_i$$

Cast-as-Intend Proof: The cast-as-intend proof ($\mathcal{P}3$) is an interactive proof that requires that the number θ, encoded in $v_j = \mathcal{E}_{pk}(\theta_j, \delta)$ of the candidate vote cv_j dubbed $YESvote$, is pledged (committed) to the voter, prior to the receipt generation. The receipt is generated in response to a challenge c generated by the voter or by some other entity.

An MP3 receipt is comprised by a concatenation of k candidate receipts $\langle \vartheta_i, \omega_i \rangle, i = 1, \ldots, k$

$$Receipt = \langle \vartheta_1, \omega_1 \rangle, \ldots, \langle \vartheta_k, \omega_k \rangle$$

$$\vartheta_i = \begin{cases} \theta & \text{if } cvote_i \text{ encrypts a } YESvote \\ 2 \cdot c - \theta \ mod \ q & \text{if } cvote_i \text{ encrypts a } NOvote \end{cases}$$

$$\omega_i = \tau_i \cdot (c - \vartheta_i) + \delta_i \ mod \ q$$

The ϑ_i part of each candidate receipt is shown to the user to verify that the chosen candidate j is the one for which ϑ_j is equal to the pledge θ. For usability reasons, only α bits of both θ and each ϑ_i values are shown to the voter. Provided that ϑ_i is well constructed, the voter is able to verify with probability $p = 1 - 2^{-\alpha}$ that the correspondent candidate vote cv_j encodes a $YESvote$. On the other hand, anyone can verify, in zero-knowledge [22], that each ϑ_i is well constructed by using the corresponding ω_i and $cvote_i = \langle u_i, v_i \rangle$ values and checking that

$$\mathcal{E}_{pk}(c, \omega_i) \stackrel{?}{=} u_i^{c - \vartheta_i} \cdot v_i \qquad (\mathcal{P}3)$$

which results directly from the definition of u_i and v_i and the homomorphic properties of exponential ElGamal [20].

Tally: MP3 votes may be counted either using a mix net or using the homomorphic properties of exponential ElGamal

$$Vote = \langle Ballot, Receipt \rangle$$

The result of the tally for each candidate i can be computed independently by performing the homomorphic addiction of each candidate vote $\mathcal{E}_{pk}(b_i, \tau)^j$ of each vote $Vote^j$. Given that b_i is either 1 or -1 the homomorphic addition is

$$\bigoplus_{j=1}^{n} \mathcal{E}_{pk}(b_i, \tau)^j = \mathcal{E}_{pk}\left(\frac{n + d_i}{2}, \varphi\right)$$

where d_i is the result for candidate j.

Chapter 3

Related Work

This chapter goes through some research to remote electronic voting solutions. Different approaches were created and refined through the years to have an electronic voting system that guarantees all the necessary requirements.

Firstly, the core properties required for each electronic voting election are detailed.

Then it is presented the initial solutions proposed by the community. Some of these approaches have been used in several systems and some of them were actually used in real binding elections.

The natural evolution of voting systems pointed to a new type, the end-to-end verifiable voting systems. These systems can achieving simultaneously voter cast-as-intended and universal counted-as-cast properties. Some voting systems are described in this chapter.

At the end of this chapter, it is presented a comparison of the studied solutions.

3.1 Properties

No existing voting system is perfect. Researchers in the electronic voting field have defined the core properties that these systems should have [23].

Accuracy: It should be not possible for a vote to be altered. After the final tally process it is not possible to eliminate a validated vote and it cannot count invalid votes;

Democracy: It is only permitted eligible voters to vote. It also ensures that eligible voters only vote once;

Privacy: Neither authorities nor anyone else can link any ballot to the voter who casts it. No one should be able to determine how any individual voted;

Verifiability: The voter should have the possibility to verify if the tally process worked successfully. For that, two verification processes can be made:

i) **Voter cast-as-intended verification** where the voter can confirm that her cast vote has the candidate who she intended to vote;

ii) **Universal counted-as-cast verifications** where the system provides to everyone a proof that the election results are the correct tally of the valid cast votes;

The particular nature of the Internet environment requires some additional characteristics for the Internet voting systems:

Coercion Resistance: A coercer cannot become convinced of how an elector votes, even if the voter cooperates with her. Of course the voter can tell a coercer how she voted, but coercion-resistance asserts that she is unable to prove it, so the coercer has no reason to believe her;

Collusion Resistance: No electoral entity or group of entities can work in a conspiracy to introduce votes or prevent others from voting;

User Friendly: The system should be easy to use by the voters. If the system is very good but the majority of the voters do not understand how to use it, the voting system will not be used;

Availability: Since the electoral process occurs in a pre-defined time, it should be accessible to all the voters from the beginning to the end of the poll;

3.2 First Main Approaches

The two main properties voting system must have are privacy and correctness. Over the years, research efforts focused on finding solutions to conciliate both properties and three main approaches emerge to solve these problems: mix-nets systems, homomorphic systems and blind signature systems.

In the following sub sections it is described these approaches combined with a voting system example.

3.2.1 Mix-Net Types

In voting systems based on mix-nets the elector's vote is decoupled from the voter's identity through a sequence of encryptions with the mixers public keys. One of the disadvantages of this system is the increase of data during the process, which leads to high computational resources requirements.

One of the most representative voting systems that uses mix-nets is CIVITAS, although most of its relevance comes from its coercion resistant properties and not from its usage of mix-nets.

The CIVITAS voting system

Probably the system with the strongest coercion-resistance set of properties is the CIVITAS system [9] that is based on the Juels *et al.* scheme (JCJ) [10]. CIVITAS is implemented in Jif, a language which enforces information-flow security policies.

The CIVITAS system allow the voter to misguide the coercer by creating a fake voting credential that will later be filtered and not processed into the election results.

The CIVITAS architecture can be seen in a general way in the Figure 3.1:

The vote protocol is separated in four steps.

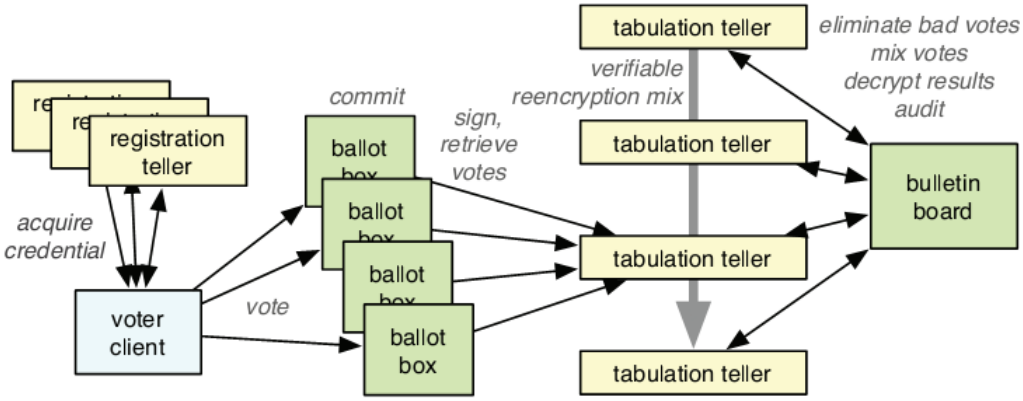


Figure 3.1: CIVITAS system's architecture

Setup: The setup phase builds the election rolls, generates the elections keys and publishes the public part in the bulletin board.

Voters Registration: Voters register to acquire their private credentials. Each registration teller authenticates a voter using the voter's registration key. The teller and voter then run a protocol, using the voter's designation key, which releases the teller's share of the voter's private credential to the voter. The voter combines all of these shares to construct a private credential.

Voting: Voting may take place immediately, or a long time after registration. To vote, the voter uses the vote client and anonymously submits her private credential and her vote, both encrypted, along with the proof of well-formed to one or more ballot boxes to guarantee availability of the vote for tabulation.

Tabulation: Before tabulation starts, all tabulation tellers retrieve the votes from each ballot box and the public credentials from the bulletin board. Then the proofs of cast ballots are verified and ballots with invalid proofs are discarded. Then duplicated votes are eliminated due to the re-voting policy. The list of submitted votes and the list of authorized credentials are then sent to a re-encryption mix net for anonymization. Finally, the decided set of countable votes is decrypted, and the corresponding decryption proofs are posted on the bulletin board.

CIVITAS provides a counted-as-cast verification. A voter may verify the correction of the tally using the information on the bulletin board, which includes zero-knowledge correctness proofs of the votes. The privacy of voter's intention is achieved using a mix-net that decouples the voter's credentials from the voter.

3.2.2 Homomorphic Type

The Homomorphic system approach was introduced in 1997 by Cramer [19]. The main idea is to allow the vote counting to process without decrypting the votes individually.

At the end of the voting phase the authorities decrypt the votes to obtain the results. The correctness in the tally process can be verified by entities.

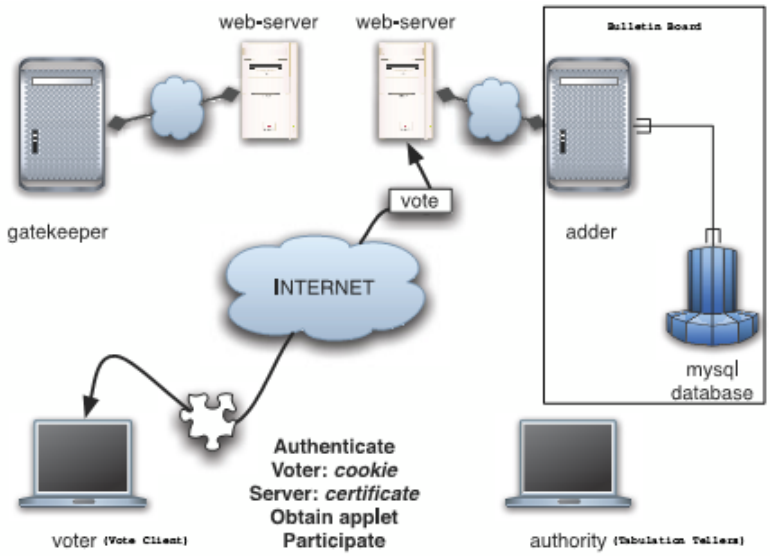


Figure 3.2: ADDER System Architecture

This method has the disadvantage of requiring high computational power and needs to provide zero-knowledge proof. Another disadvantage is the limitation of election's answer type, since only accepts Yes / No answers.

The ADDER voting protocol based on the homomorphic method is presented below.

The ADDER voting system

The ADDER was introduced by Aggelos Kiayias *et al.* in 2006[24]. This protocol was developed in Java.

The ADDER is an homomorphic based voting system which presents the following architecture:

The system is comprised by three phases:

Setup: The gatekeeper configures the system and the administrators starts the process with the creation of the public cryptographic parameters and the relevant information. Then each tabulation tellers creates her own asymmetric key pair based on the election parameters and the public part is published in the bulletin board. At the end of this phase, all the tabulation tellers gathers to generate a shared election key pair and publish the public part in the bulletin board.

Voting: After voting with the client vote, the vote is encrypted with the election public key and sent to the bulletin board with the correspondent zero-knowledge proof.

Vote Count: At the end of the election, all the votes are homomorphically aggregated and the election result is published. All the tabulation tellers cooperatively decrypt the votes aggregated encryption. The result decryption and the proof of correct decryption are published in the bulletin board. Everyone can then verify the correctness of the homomorphic vote count.

The ADDER guarantees voters privacy due to the homomorphic approach, provides universal counted-as-cast verification but has no cast-as-intended verification. This voting system does not provide any

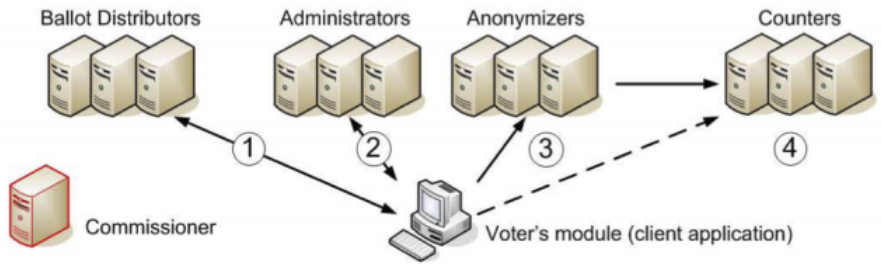


Figure 3.3: REVS System Architecture

coercion-resistance and has the weak assumption that the client must trust the client vote. Still it is an important voting system for the SM&V due to the homomorphic method.

3.2.3 Blind Signature Type

The Blind Signature method was first used in 1993 by Fujioka [25] to solve the vote validation problem without decreasing privacy. The main idea is to decouple the vote from the identity. It has the disadvantage of needing the voter to be active in two phases to guarantee the correct function of the protocol.

The REVS is an example of a voting system based on the blind signature.

The REVS voting system

The REVS voting system was introduced in 2003 by Rui Joaquim *et al.* [26].

It is a blind signature based remote Internet voting system with focus on robustness and availability. Hence REVS has no single points of failure as it supports the replication off all online servers.

The REVS architecture is presented in Figure 3.3.

The vote protocol is composed by three phases:

Setup: A commissioner setups the election information, then it is signed with the commissioner's private key and then distributed to the election servers.

Voting: With the vote client, the elector gets the ballot from a ballot distributor and enters her voting choice. The vote client adds a unique random identifier to the vote and validates it by collecting at least half plus one administrator's blind signatures on the vote. The blinded signatures are then unblinded. The vote and signatures are encrypted with the election public key and sent to a tabulation teller through an anonymizer.

Vote Count: This phase starts with the revelation of the election private key to the tabulation tellers. Each tabulation teller decrypts the received votes and publishes the results. At the end the results are gathered, the duplicated votes are removed and the final results are published.

The REVS allows the voter to verify that her vote was or not modified by the vote client and if it is included in the final tally. This verification is possible because each vote has a unique random identifier.

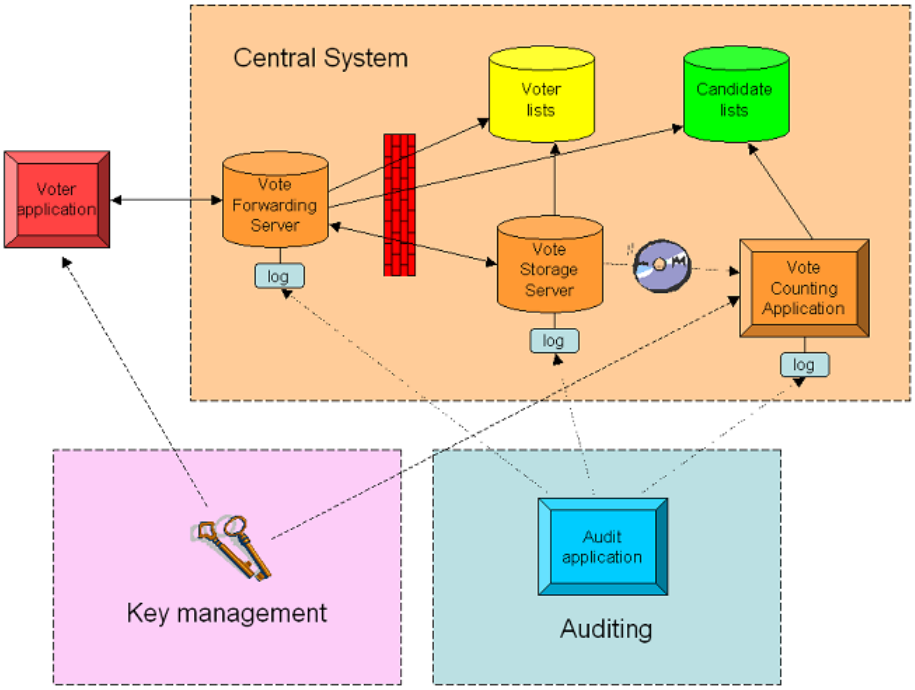


Figure 3.4: Estonian System Architecture

This protocol protects the voter's privacy as long as the voters are correctly anonymized. The REVS has the weak assumption that the voter has to trust the client application.

3.3 Real binding elections

There are many voting systems but only few are used in real biding elections.

Next it is described two voting systems that are already in use: the Estonian and Norwegian voting Systems.

3.3.1 The Estonian voting system

The Estonian e-Voting system was first used in 2005 [27]. The solution was not proposed as a substitute for the traditional vote, but as a complement. The voter may vote electronically before the election day, and may cancel that vote and vote again traditionally on the election day. The last numbers of the utilization are from 2011 where 24.3% of the votes were made in internet, representing 56.4% of all votes made in advance.

The system feature several authentication methods, namely: *i)* ID-Card, the national identification card; *ii)* digi-ID, a digital identification only used for electronic environments *iii)* mobile-ID, which allows authentication with the voter's mobile phone, but the voter still have to vote on a computer with Internet connection.

The Estonian system architecture is presented in Figure 3.4.

The voting protocol is composed by four steps:

Setup: First, the candidate list, the electoral rolls and the voter's public keys are stored in the vote forward server. The electoral key pairs are generated and the public part is published.

Voting: The voter starts the voter client application which connects to the vote forwarding server. The voter authenticates with one of the three possible ways described above and then the vote forwarding server sends the ballot to the voter. After voting, the voter client encrypts the vote with the election public key and requests a signature from the voter's card. At the end, it sends the encrypted vote to the vote forward server who redirects to the vote storage server. The voter can vote several times, but only the last one is considered for the tally.

Vote Counting: This phase only starts when the electoral voting period is over. The physical votes are joined with the vote storage servers and creates a list of encrypted votes contained only by the last votes of the electors who did not vote at the poll site. The information is transferred physically (e.g. a CD) to the vote counting application. First the key holders enter their private key shares. Then the votes are decrypted and published the final results together with a list compound by the vote's hash for audition purpose. The vote count method is "black box". To minimize this transparency issue the Estonian voting system allows the re-counting of the votes by different vote counting applications.

The Estonian voting system do not guarantee cast-as-intended property since the voter has no way to detect what did the application really voted too. The counting mechanism is also not verifiable.

In terms of privacy it is possible to create a vote receipt, but only the insiders or the system auditor can use this information. One of the disadvantages is the possibility to violate the privacy if the counting application has access to the vote storage server content.

3.3.2 The Norwegian voting system

The Norwegian e-Voting system architecture described in [28] was used in 2011 in a voting trial local elections [4]. In these trials 26.4% of the votes were made by Internet which represented 72.5% of advanced votes.

The Norway system architecture is presented in Figure 3.5.

The voting protocol is separated in three phases: the setup, the voting and the vote counting phase.

Setup: The electoral board starts generating the key pairs to the ballot box and the receipt generator. Then the electoral board creates a shared electoral key pair among the tabulation tellers. After this process, the electoral board generates the ballot verification codes and sends to each voter. Some partial information of vote verification code is given to the ballot box so it can start the computational process when receive the information of the vote.

Voting: To vote, the user votes in the intended candidate. The system encrypts the vote with the electoral public key, it signs the vote with her private key and sends it to the ballot box. After

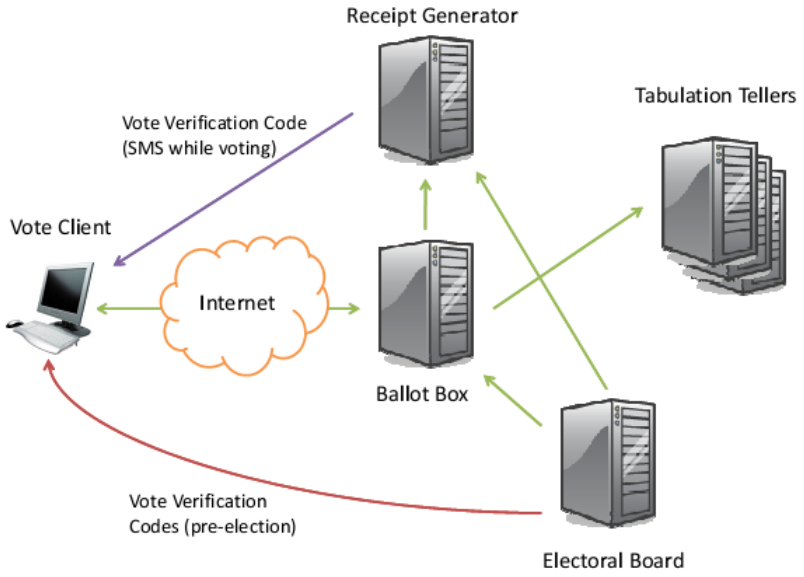


Figure 3.5: Norwegian System Architecture

receiving the information, the ballot box transforms the encryption of the candidate identifier into the encryption of the verification code and transforms again to be under the receipt generator public key. The receipt generator receives the transformation, performed previously and after decrypting, sends it back to the voter via an independent channel, e.g. a text message to the voter's mobile phone with the purpose to check that the verification code corresponds to the voter's candidate choice.

Vote Counting: The votes are mixed in the ballot box by a verifiable mix net. After, they are sent to the tabulation tellers and cooperatively they are decrypted in a verifiable way so that everyone can verify the election results.

The vote is signed by the elector so the ballot box and the receipt generator cannot modify the voter's choice. But if the voter's client, the ballot box and the receipt generator are in collusion, it is possible to change the vote without voter detection.

The tally process is performed by a verifiable mix-net. Due to the coercion issue the system do not achieve the universal counted-as-cast verification property.

3.4 End-to-end verifiable voting systems

An end-to-end verifiable voting system is a system that provides universal counted-as-cast verification and voter cast-as-intended verification. This property is imperative to remove the untrustworthiness of the voting system implementation. Given that the system is verifiable end-to-end, the way that it is implemented is not relevant, much in the same way that the routing of packets with a virtual private network is mostly irrelevant (provided that they arrive) for the security of the channel.

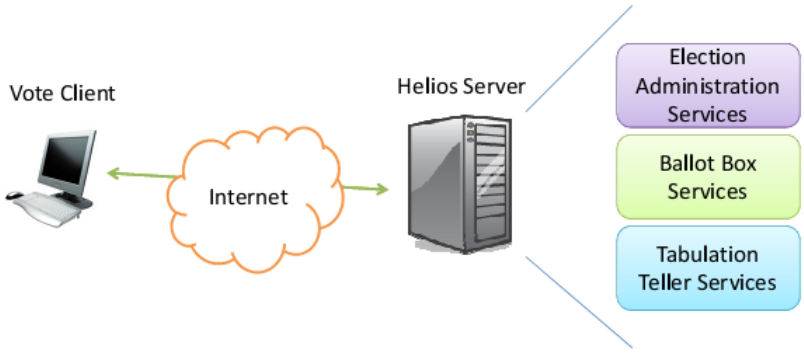


Figure 3.6: Helios System Architecture

There are several voting systems that are end-to-end verifiable, such as: Helios [29], Scratch Click & Vote [30], EVIV [1] and VeryVote [31].

Both Helios and EVIV voting systems are described next. The Helios is one of the most used systems in this subject and EVIV is also mentioned since it was the base to create the SM&V voting system.

3.4.1 The Helios voting system

The Helios voting system is currently at version 3.0 [29] and is the first Internet voting system that provides cast-as-intended verification [32] based on the Benaloh's "cast-or-verify" method [33] and counted-as-cast verification, which is supported by the homomorphic vote count.

The Helios architecture is presented in Figure 3.6.

The vote protocol is divided in three phases: the setup, the voting and the vote counting phase.

Setup: The trustees generates a shared election key pair and publishes the election public key. After that, the election administrator starts the election by sending the election link to the voters by email.

Voting: The voters open the link inside the email and download the application for voting. Then, they choose their candidate and confirm the selection. The vote is encrypted and a hash of her choice is given to the voter. Now the voter can cast or audit her vote. If she casts the vote, it is necessary to enter the username and the password and the vote is cast. When the Helios servers receives the vote, it responds with the vote hash value by email. If she chooses to audit the vote, the client vote reveals the randomness in the vote encryption, so it can be verified by the voter with a vote verification software that must be trusted.

Vote Counting: The final step is sending the votes to the tabulation tellers and the encrypted votes are counted since they have homomorphic properties. The election results are published next to the proof of correct decryption.

With the vote verification software, the Helios system has a cast-as-intended soundness of $1/n$ where n is the number of verified vote encryptions.

There are some possible attacks to Helios system exploring the voter client using browser rootkit, but they can be prevented by using independent verification tools, ideally from another computer.

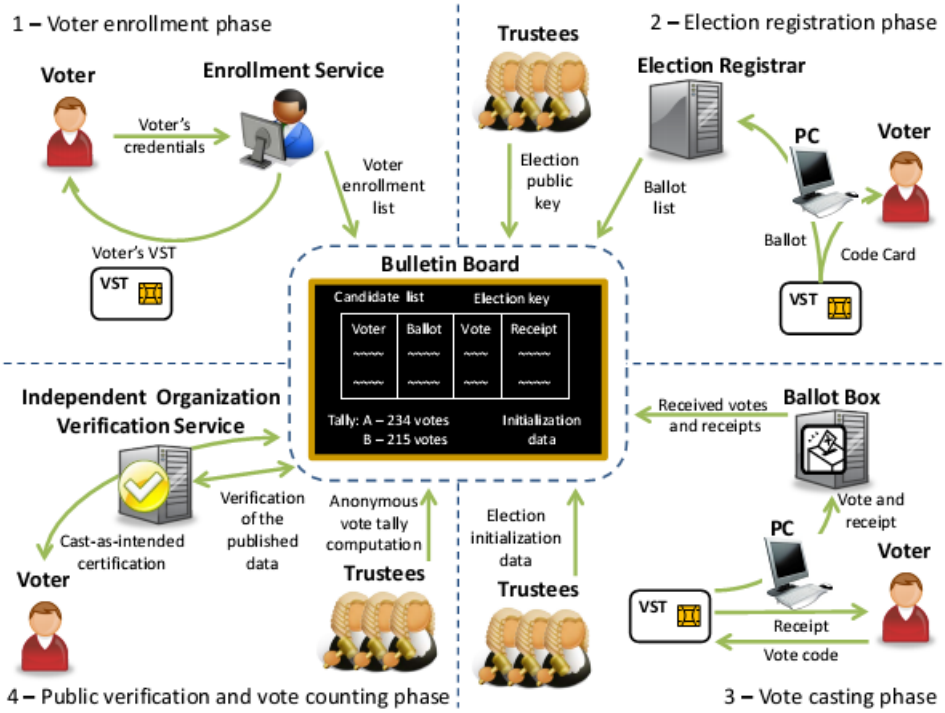


Figure 3.7: EVIV System Architecture [1]

In terms of privacy the voter must trust the voter client to guarantee that it will not reveal the randomness used in the vote encryption.

3.4.2 The EVIV voting system

EVIV: an end-to-end Verifiable Internet Voting system was introduced in 2012 by Rui Joaquim, Carlos Ribeiro and Paulo Ferreira [1].

The EVIV system guarantees strong authentication and privacy together with the possibility to vote in public places. The majority of the voting systems assume that the voter's client device must be trusted by the voter, which nowadays is a dangerous assumption. Like Helios, the EVIV system assumes that the voter's device is not trusted. This gives a full voter's mobility because the elector can vote wherever she wants to, independently of the trustworthiness of the device.

The EVIV architecture is presented in Figure 3.7

The EVIV protocol is separated in four main phases:

Voter Enrollment: The process starts with the voter identifying herself with her credentials to the EC and then receiving a Voter Security Token (VST) with the private key and the voter's credentials. After this process, EC publishes the public information about the enrollment in Bulletin Board (BB).

Election Registration: The election registration is needed for each election that the voter wants to participate in. The registration phase can start some months previously to the election and end in the beginning of the voting phase.

First it starts with the generation of a threshold election key pair among a set of n trustees. To

decrypt the message the collaboration of a subset of t trustees is needed ($n > t$). The key pair will be used to generate the vote ballots and to decrypt the votes for the tally process.

Then the electoral commission publishes the public election key in the bulletin board combined with the list of candidates. After publishing, the voters have to register in the election. This step is divided in two separated sub steps:

1. **Ballot creation:**
   (a) The voter establishes a secure connection between the voter's device and the Election Register (ER) with the VST. Both have to authenticate;
   (b) The election public key and the candidate list are downloaded from the bulletin board to the VST through the ER;
   (c) The VST generates the ballot to the voter using the MarkPledge 3 technique (Section 2.7) signs it with voter's private key, records the result in the VST and sends it to the ER;
   (d) The ER verifies the correctness of the cyphered ballot and publishes on the BB.
2. **Generation of voting codes:**
   (a) After the cyphered ballot is published, the VST generates the Code Card (CC) that represents a list of random characters, each corresponding to a different candidate. The VST also generates a confirmation code that is used for the voter's cast-as-intended verification;
   (b) The CC is printed through the voter's device.

Voting: The voting phase starts with the generation and publication of a random number (as a challenge). This generation is made in a distributed way by the trustees, and it is assumed that at least one trustee is honest so that the challenge is in fact random. Then, the information is published in the BB so that the Ballot Box (BBox) can start receiving votes.

The voting can be done wherever the voters want to vote. The protocol was made so that despite voting in a non-protected device, the privacy is still guaranteed. Only a card reader is needed in the device.

The voting phase is divided in 4 steps:

1. The voter connects the VST via card reader to the computer and establishes a secure connection through the Internet, authenticating the VST;
2. The initialization data and the list of candidates are downloaded from the BBox, presenting the list of candidates to the voter;
3. The voter contacts the VST informing the voter's choice, and then the VST encrypts the vote as shown in Figure 3.8. The cyphered vote is generated from the necessary rotations from the YESVote on the ballot to the candidate choice and both the encrypted vote and the receipt are signed with the VST private key. The receipt is shown to the voter via her device.;

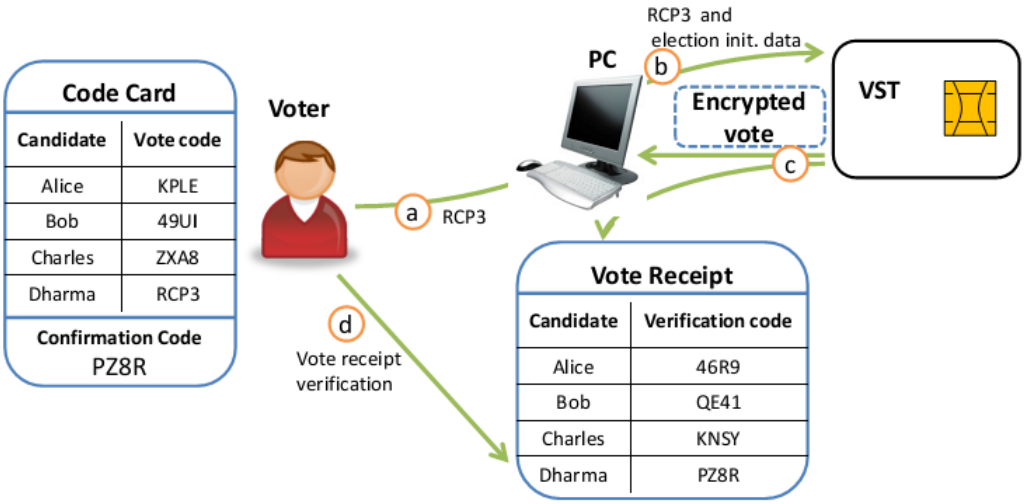


Figure 3.8: Candidate selection and receipt verification. In this example the Dharma candidate is chosen with the vote code RCP3. The voter can verify her selection in the receipt comparing with the confirmation code PZ8R.

4. The encrypted vote and the receipt are sent to the BBox. Both are verified by the BBox before accepting them.

At the end of the voting phase, the EC validates the signs of all information received by BBox and then publishes it to the BB.

Vote Counting and Public Verification: Before the counting process starts, data is verified in terms of correctness and validity. After this first verification, vote counting can start and finishes with a final verification of counting process. These steps are detailed below:

Election Data Verification: This step is separated in three different tasks:

1. The ballot needs to guarantee if one and only one candidate was selected and if is signed by the voter's VST;
2. Verifies if the encrypted vote was created from the registration's ballot and if the receipt was created from the encrypted vote and the initialization data. The Independent Organizations (IO) can verify the vote and receipt signature;
3. Cast-as-intended verification from the voters. This task can be done by the Verification Service (VeryServ) from an IO. The voter can verify that the confirmation code in the CC, corresponds to the verification code in her receipt. If an anomaly is detected, the correspondent vote is abrogated and the elector can vote again. All these changes have to be done before the vote counting process starts. None of the encrypted votes are revealed so that privacy is guaranteed to all voters.

Vote Tally: To provide universal counted-as-cast verification, the encrypted votes have homomorphic properties, and the process is separated in three tasks:

1. BBox aggregates in an homomorphic way all the valid encrypted votes (the others are previously removed);
2. The aggregated encrypted votes are decrypted by at least t trustees by and algorithm of threshold decryption (Section 2.5), preserving the voters privacy because none of the votes are decrypted individually. With this process, the election results and the proof of the decryption correctness are obtained;
3. Both the election results and the decryption proofs are validated by the EC and sent to the BB.

Vote Tally Verification: In the final step, each IO can verify the tally by processing themselves their own homomorphic tally. The IO can verify the correctness of the results by the decryption proofs. Anyone with sufficient knowledge and computation power can verify all the election data. All the verification is based only on public information published in the BB.

The EVIV system is founded on the following assumptions:

- MarkPledge 3 does not fail;
- There is at least one honest trustee;
- In the verification process and the tally process there must be at least one honest entity;
- The voter uses an honest vote verification entity;
- There is no collusion in more than t out of n trustees;
- The VST does not disclose the voter's choice;
- Only the VST and the voter knows the voting codes.

EVIV guarantees total mobility, privacy and integrity to the electoral process, but is poor in coercion-resistance. SM&V is a variant of the EVIV voting protocol with coercion-resistance properties.

3.5 Voting Systems Comparison

An overall view of some important properties related to the privacy and verifiability are summarized in Table 3.1.

Generally, the vote client knows the voter's choice, the exception is the EVIV vote protocol which merge a code voting technique with the MarkPledge vote encryption. The use of code voting protects the voter's privacy from a malicious vote client.

The voter's privacy can be broken by collusion of two or more election servers. In the REVS system the privacy breaks depending on the number of anonymizers. In the Estonian system if the counting application has access to the vote storage server content, the privacy is broken. The Norwegian system

	Vote client knows vote	Shared election key	Collusion can break privacy	Vote client can create receipt	Universal Counted-as-cast Verification	Cast-as-Intended Verification	Collusion can add abstaining voters	Coercion Resistance
CIVITAS	Yes	Yes	No	No	Mix-Nets	No	Registration tellers	Yes (Fake Credentials)
ADDER	Yes	Yes	No	Yes	Homomorphic	No	Yes (Detectable by voters)	No
REVS	Yes	No	Anonymizers	Yes	No	No	Administrators (Detectable by voters)	No
Estonian	Yes	Yes	Vote couting application with access to the list of votes	Yes (only auditors could use that information)	No	No	No	No
Norwegian	Yes	Yes	Ballot box + Receipt generator + Voter's client	Yes (only auditors could use that information)	No (Mix-Nets verification only by auditors)	Yes	No	No
Helios	Yes	Yes	No	Yes	Homomorphic	Yes	Yes (Detectable by voters)	No
EVIV	No	Yes	No	No (VST)	Homomorphic	Yes	No	No

Table 3.1: Internet Voting Systems comparison

requires collusion between the ballot box, the receipt generator and the voter's client. In the rest of the voting systems, the privacy can only be broken by collusion of the election key holders.

Almost all the voting protocols that use the vote client to encrypt the vote, creates a receipt that can prove the voters choice, with the exception of CIVITAS and EVIV systems. In the cases where the system do not provide count-as-cast verification, i.e. the Estonian and Norwegian systems, the vote receipt can only be used by insiders or by auditors. The CIVITAS system uses zero-knowledge process to select the valid votes, thus the original vote encryption does not constitute a vote receipt. In EVIV system the receipt is created in the VST card.

Only three systems do not provide universal counted-as-cast verification. The REVS system only provide individual voter verification and the Estonian system a procedure to verify the final result correctness. The Norwegian voting system uses a verifiable mix-net tally but only auditors can verify the process.

To provide voter cast-as-intended verification, the system must have verifiability at vote client side and server side. *i)* In the client side, the Norwegian system use code voting mechanism, while the EVIV system use both the code voting with the MarkPledge vote encryption and verification encryption. The main advantage in this technique consists in the fact that the vote client does not know which candidate was selected by the voter. The Helios system uses Benaloh's cast-or-verify mechanism to check if the vote client is creating the correct vote encryption. *ii)* In the server side, it is important to ensure that it is not possible to any collusion of the server to modify the voter's vote. In the Norwegian and EVIV systems the vote is digitally signed. In the Helios voting system, it is possible a collusion of election servers, however it can be detected by the voter.

In almost every system it is very hard to add votes for the abstaining voters since they need to be digitally signed. In the ADDER, REVS and Helios systems it is possible to add votes but it is easily detected by looking at the registry of the voters who cast a vote in the election. The CIVITAS system is the exception, because this system does not record who voted in the election, so a collusion of the

registration tellers can insert votes for the abstaining voters.

One of the main desired properties in the remote electronic voting systems is to be coercion resistant. Only one of the studied voting systems achieves this property, the CIVITAS system. The coercion resistance property is provided by giving the possibility to construct a fake credential that will be used when casting the vote.

In conclusion the real binding scenario voting systems like Estonian does not offer sufficient privacy and verification guarantees to voters. However the Norwegian system presents a solution to prevent vote manipulation, but due to coercion issues, the voters still must trust the election authorities and the auditors for the election integrity. In terms of privacy in both solutions it is only needed the collusion of election servers to break the vote's anonymity. Analyzing and comparing all the information gathered, it is conclusive that the EVIV system is a combination of the best properties of the previous voting systems, however it has one flaw that cannot be overlooked, it has no coercion resistance.

Chapter 4

Architecture and Trust Model

Scroll, Match & Vote (SM&V) is an end-to-end verifiable and n-collusion-resistant Internet voting system that attains also coercion resistance (Section 3.1) at the expense of reduced mobility when compared with EVIV.

SM&V proposes a new voting interface with coercion resistant properties combined with a variant of the EVIV voting protocol [1].

With the SM&V election system, the voter may cast her vote anywhere using nothing more but her smartphone and an Internet connection, however, prior to election day, in a period that may be of a few minutes to a few months before the election, the voter must register for that election in a private booth. The privacy of this registration is paramount for the coercion resistant property of the voting system.

Besides the registration and voting phases there is yet another phase that every voter should do; the verification phase. The verification phase takes place any time after the voting phase with the purpose of ensuring that the casted vote is, in fact, counted as cast.

From a voter's perspective the voting machine is her smartphone, although as it is described below, the actual ballot creation is performed by an applet running inside a UICC, a secure Secure Digital Card (SDCard) or any other Secure Element (SE) inside the phone.

4.1 Elements of the model

Besides the voter and her smartphone, there are six other types of entities participating in the system.

The **Bulletin Board** ($\mathcal{BB}$) is the service responsible for the publication of all election public data. The data published cannot be deleted and it is always authenticated, i.e. digitally signed.

The **Trustee** service is run by n different entities. The Trustee service exists in order to share the control over the voter's privacy and the election's integrity among several entities (the trustees). The trustees can be the political parties and/or any other authorized entity (e.g. an election observer or a non governmental organization).

The **Electoral Commission** (EC) service is responsible for the entire electoral process; namely, the EC is responsible for the voters enrollment system, the actual voting system and the authentication of

all election public data.

The **Helper Organization** (HO) are entities that run services of vote verification by using public information published by the voting system. They can display to the voter her individual vote/receipt pairs. They also provide an application to generate the 2D-codes needed for the registration.

The **Pledge Display Device** (PDD) only purpose is build an untappable channel between the voter and the SE, to transport a small secret code to the voter: the "pledge".

Finally, the **Voting Machine** application that creates both the votes and the receipts is run by a secure element (*SE*) inside the smartphone of the voter, e.g. the UICC (Universal Integrated Circuit Card).

4.2 Properties and Trust Model

The proposed system exhibits a number of security properties under a specific number of assumptions. Some of the assumptions are necessary to ensure integrity-related properties, and others are needed to ensure confidentiality-related properties. Below, we enumerate some of the most relevant properties and the assumptions under which they are achieved, and also some relevant limitations, i.e. relevant security properties that the proposed solution does not have. The assumptions of the system are the properties required from the environment where the system is to be deployed, therefore we also discuss how they may be achieved and how reasonable they are.

4.2.1 Integrity Properties

P$_{I1}$ - No votes can be added, deleted or modified without detection.

P$_{I2}$ - Every vote is counted-as-recorded.

P$_{I3}$ - Every voter can verify that her vote is recorded-as-intended with a soundness of $(1-2^{-\alpha})^{\rho.(n_c-1)}$.[1]

These properties are ensured under the following assumptions:

A$_{I1}$ - The data published in the $\mathcal{BB}$ cannot be deleted and it is always authenticated, i.e. digitally signed.

A$_{I2}$ - There is no collusion of more than $t < n$ out of n trustees, where t and n are configurable security parameters.

A$_{I3}$ - At least one honest organization or entity with cryptographic capabilities will verify the correctness of all the data published in $\mathcal{BB}$.

A$_{I4}$ - The SE and the coercer do not collude.

Assumption A$_{I1}$ is a common assumption of e-voting systems therefore some proposed web bulletin boards fit SM&V requirements [34]. The reasonableness of assumptions A$_{I2}$ and A$_{I3}$ depend on the number of available trustees and helper organizations. If there are enough trustees and helper organizations there is a high probability that there are no more than t faulty trustees and there is at least one

[1] The protocol security parameters ρ and α are discussed in Section 5. n_c is the number of running candidates.

honest helper organization. Given that, SM&V trustees and helper organizations may be run by entities with no specific availability of performance requirements (cf. section 5), it is easy to assume that there will be a reasonable number of them available. Assumption A_{I4} means that SM&V may not defeat a coercer which is able to simultaneously coerce the voter and tamper her secure element. This assumption is, nevertheless, much weaker than other coercion resistant assumptions. In JCJ/Civitas the whole vote machine is required to be trustworthy, independently that the voter is being coerced or not, while in SM&V only a small fraction of the voting machine is required to be trustworthy – the secure element, everything else, including the interface with the user and the communication with the voting machine do not need to be trusted.

4.2.2 Confidentiality properties

P_{C1} - No one but the voter and her *SE* knows the voter's chosen candidate.

P_{C2} - Coercion Resistant: Voters cannot prove how they voted, even if they can interact with the adversary while voting.

These properties are ensured under the following assumptions:

A_{C1} - The SE (which performs the vote encryption) does not disclose the voters' vote choices.

A_{C2} - Neither the SE or the PDD disclose the "pledge" to anyone but the voter.

A_{C3} - Only legitimate registration precincts will possess certified PDDs, i.e. PDDs with a certificate signed by the election committee for that specific election with that specific validity.

A_{C4} - The channel between the PDD and the voter cannot be tapped.

Assumptions A_{C1} and A_{C2} requires trustworthy SEs. SEs are secure by design. Although some exhibit vulnerabilities [35], it is far easier to build a trustworthy tailor-made SE than a trustworthy tailor-made voting machine with all the user interfaces and communication devices. A_{C2} requires that PDDs do not disclose the "pledges" to any one but the voter, which may be achieved, at some level, by removing any communication interfaces with the exception of the one used to communicate with the SE, e.g. NFC. Assumption A_{C3} requires that PDD's certificates are issued with the identifier of the election for which they are being deployed and that each PDD is deployed on a valid registration booth. A_{C4} is one of the most complex assumptions of the protocol to satisfy by the environment; any one with a camera is able to record and transmit what is being displayed by the PDD within the voting booth. However, this is a common assumption of most voting protocols, including the classical paper-based voting.

Please refer to Appendix A for the proofs of SM&V properties under the above assumptions.

4.2.3 Relevant Non Properties

Force-Abstention - An attacker may obtain a proof of abstention by looking at the tally and verifying if there is a vote for the coerced voter, Therefore anyone may force a voter to abstain and then verify

if she complied. The voter may however vote physically without being detected by the coercer, given that any physical vote overrides e-votes.

Randomization - An attacker may force a voter to vote randomly, preventing the voter from voting on the chosen candidate. However, again, the voter may still vote physically given that any physical vote overrides e-votes.

Pre-attack surveillance - A coercer may learn with some probability the "pledge" of a voter by checking the $\mathcal{BB}$ and learning the code next to the probable candidate choice of the voter. After learning the "pledge" the coercer may force the voter to revote on another candidate. The coercer does not know, however, for sure, if the learned "pledge" is the correct "pledge". This vulnerability is shared with Civitas [9]. Again, in SM&V, the voter may evade the coercion by voting physically and replace her e-vote.

4.3 SM&V Usage Scenario

In a general view SM&V can be presented in figure 4.1. From a conceptual perspective the architecture in SM&V is similar to the EVIV system since it is a variant of it. The main differences in the process are described in the next sub section.

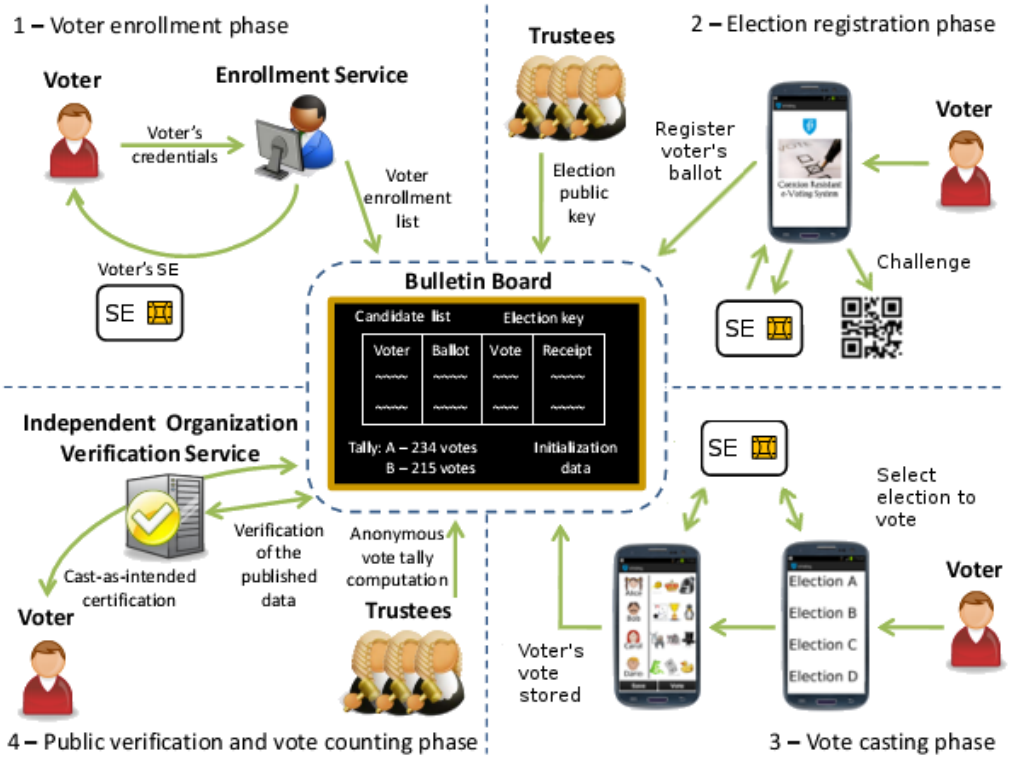


Figure 4.1: General view of SM&V.

4.3.1 Registration

The registration phase is the most complex phase of the voting process. It starts after a set of trustees generate the public key of the election and ends just before the vote casting, i.e. it can be made even on election day.

Prior to registration the voter should generate and print two 2D-codes and take it with her for the registration. The 2D-codes may be generated by the user, her self, by an online helper organization or even by a coercer, provided that he is not colluding with the SE (check assumptions in section 4.2). One of the 2D-codes (the second one to be used) contains a random number which is used to challenge the vote machine, and prevent it from generating a compromised ballot. The first 2D-code is a commitment to the second to prevent the voter from leaking the "pledge" to the coercer (cf. section 5.3). For usability purposes the two 2D-codes should be of different types (e.g. a PDF417 and a QR-code).

To register, the voter should take her smartphone to a private booth, specially prepared for the purpose, and press register on her smartphone voting application (Figure 4.2, step a). She will then be asked to: i) choose the election, ii) read one of the 2D-codes with her smartphone camera, iii) tap her phone against a special device dubbed "Pledge Display Device" (PDD), whose only purpose is build an untappable channel between the voter and the SE, to transport a small secret code to the voter: the "pledge".

The PDD owes its existence to the untrustworthiness of the voter's smartphone. Being a multipurpose device with many different running applications it is assumed that anything displayed on its screen may be leaked to a coercer. The PDD's only purpose is to receive, decrypt and display the "pledge". It does not know anything else about the voter therefore it cannot compromise the voter privacy. Still, to ensure that there is no possibility of using a false PDD to display the "pledge", only certified PDDs with a specific certificate signed by the electoral commission may be able to decrypt the "pledge" (cf. section 5.3).

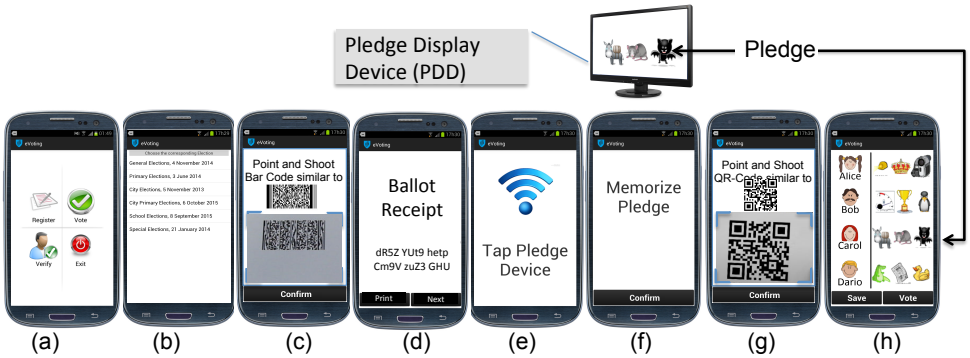


Figure 4.2: Registration procedure.

After tapping the smartphone on the PDD, the "pledge" appears on the PDD's screen and the voter is asked to memorize it. The pledge is a random number, generated by the vote machine, which is shown on the screen and encoded into a set of symbols (cf. section 7.1). After confirming seeing the pledge, the voter is asked to read the second 2D-code using her smartphone's camera. Once the 2D-code is read, the smartphone displays the ballot to the voter. The ballot is shown as two scrollable lists side by side.

The list on the left contains the names of the candidates, while the list on the right contains an equal number of sequences of symbols, one of which is the “pledge” shown in the PDD. To prevent coercion the voter should also memorize a few other sequences of symbols to be used as false “pledges” in case of coercion. The registration ends either by saving the generated ballot or by engaging immediately in the voting phase.

4.3.2 Voting

The voting phase starts with the same screen shown at the end of the registration. In fact, registration and voting may be done in a single sequence of steps at election day, or in two distinct sequences on separate occasions. Voting is accomplished by sliding one or both lists in the screen such that the chosen candidate and the sequence of symbols with the “pledge” become aligned (they can be visible or not, provided that they are aligned), and press VOTE. Anyone next to the user will not be able to tell in which candidate is the voter voting without knowing the “pledge”. Given that the voter is able to lie to the coercer about the sequence encoding the “pledge”, even a coercer will not be able to tell in which candidate the voter is voting.

4.3.3 Verification

After Voting the user should check if her vote was counted as intended by verifying that: her vote is in the poll, the 2D codes published match the printed ones, and that the vote is counted for the chosen candidate. The verification can be done using the mobile voting application, but it is recommended that the voter uses another Internet device with a simple web browser connected to a Helper Organization of her trust. Helper Organizations (HO) are entities that run services of vote verification by using public information published by the voting system. Anyone with enough computer power might create a HO provided that it is trustworthy for some of the voters.

The task of a HO is to run a complete cryptographic check on all the votes and redo the vote tally to verify the overall result (cf. section 5.5). After verifying the correct construction of the ballot and vote the HO is able to display to the voter her individual vote/receipt pairs. The vote/receipt pair shown to the voter is similar to the voting screen shown to the voter, but without the ability to change the matching between the candidates and the sequences of symbols. The voter will then verify that the the code seen of the PDD display, i.e. the “pledge” appears next to her chosen candidate. If the voter verifies that her vote is not correctly cast, she may cast another vote on the Internet, or go to a vote precinct if that option is available.

Chapter 5

Scroll, Match and Vote (SM&V)

The SM&V election protocol has five phases: Voter Enrollment, Election Preparation, Ballot Registration, Vote Casting, and Vote Verification and Counting. The following sections describe the communication steps carried on each of these phases in detail. Each step is identified by an expression of the type $X \rightarrow Y : M$, where X is the sender, Y the receiver, M the message, and $\rightarrow$ stands for either an NFC communication [36], an OTA communication or an interprocess communication within the smartphone. The expression $\langle M \rangle_{sk_B}$ stands for message M signed by B, while $\langle M \rangle_{pk_B}$ stands for message M encrypted for B and $\{n_j\}_{j=1}^{k}$ stands for a set with every element n_1 to n_k. It is assumed that the voter's smartphone is able to communicate through NFC with the PDD. NFC communication starts only when two NFC-enabled devices get almost in touch with each other (dubbed tap each other) and, at least, one of them had queued a message to be sent by NFC.

5.1 Voter Enrollment

Voter enrollment takes place only once per voter and is valid for several elections.

1. $\mathcal{V}_i \rightarrow EC$: Voter Credentials

   Voter enrollment starts when the voter $\mathcal{V}_i$ provides the Electoral Commission her credentials proving to be a valid voter.

2. $\begin{cases} EC \rightarrow \mathcal{V}_i : \text{SDCard with Voting Machine and the Voter Electoral Credentials} \\ EC \rightarrow TSM \rightarrow UICC_i : \text{Upload Voting Machine and the Voter Electoral Credentials} \end{cases}$

   In response, the Electoral Commission either provides the voter with a secure SDCard containing the voting machine and the voter credentials to be used for voting, or uploads the voting machine software to the voter's UICC, along with the voter electronic electoral credentials, through a OTA Trusted Service Manager (TSM) [37]. The former option is simpler and more secure but requires delivering a physical card to each voter.

3. Market/Store $\rightarrow$ Smartphone : Voting Application

Finally, the enrollment phase ends by downloading the voting application to the voter's smartphone from the voter's preferred AppStore. Notice that the voting application and the voting machine are different entities. The former handles the user interface while the latter handles the actual vote encryption.

5.2 Election Preparation

The election preparation phase takes place once per election and is performed by the Electoral Commission, the Election Trustees and the $\mathcal{BB}$.

1. $EC \rightarrow \mathcal{BB} : \langle electionParameters, \mathcal{C} \rangle_{sk_{EC}}$

   The election preparation phase starts with the Electoral Commission publishing on the Bulletin Board the election candidate list $\mathcal{C}$ and the public election parameters, such as:

   - Election key pair parameters;
   - List of candidates;
   - Max registration attempts;
   - Max challenge creation attempts;
   - Alpha Bits (Least bits to display verification code)

2. $\mathcal{T} \rightarrow \mathcal{BB} : \langle keyGenerationData,\ pk_{\mathcal{T}} \rangle_{sk_{\mathcal{T}}}$

   The second step in the election registration phase is the creation of a shared threshold ElGamal election key pair by the set of Trustees $\mathcal{T}$. In [38, 39] the reader can find more details on how to create a (t, n)-threshold election key pair, for the ElGamal cryptosystem, and how to decrypt a message using the shared private key. The input messages (cryptographic key parameters), the public outputs of the key generation protocol and the election public key ($pk_{\mathcal{T}}$) are all published in the public Bulletin Board. Each trustee signs her messages before sending them to the Bulletin Board.

3. $EC \rightarrow \mathcal{BB} : \langle id_{election}, pk_{\mathcal{T}}, \mathcal{C} \rangle_{sk_{EC}}$

   The Electoral Commission verifies the election public key generation data, published by the Trustees, and validates it by signing together with the candidate list $\mathcal{C}$ and the election identifier (which is a generic identifier), and publishing the signed tuple on the Bulletin Board.

5.3 Ballot Registration

Each voter may register one or several ballots for each election, although only the last one may be used for voting. There are five entities involved in the registration phase: the voter, the voter's smartphone, the SE, the PDD, and the $\mathcal{BB}$ (Figure 5.1).

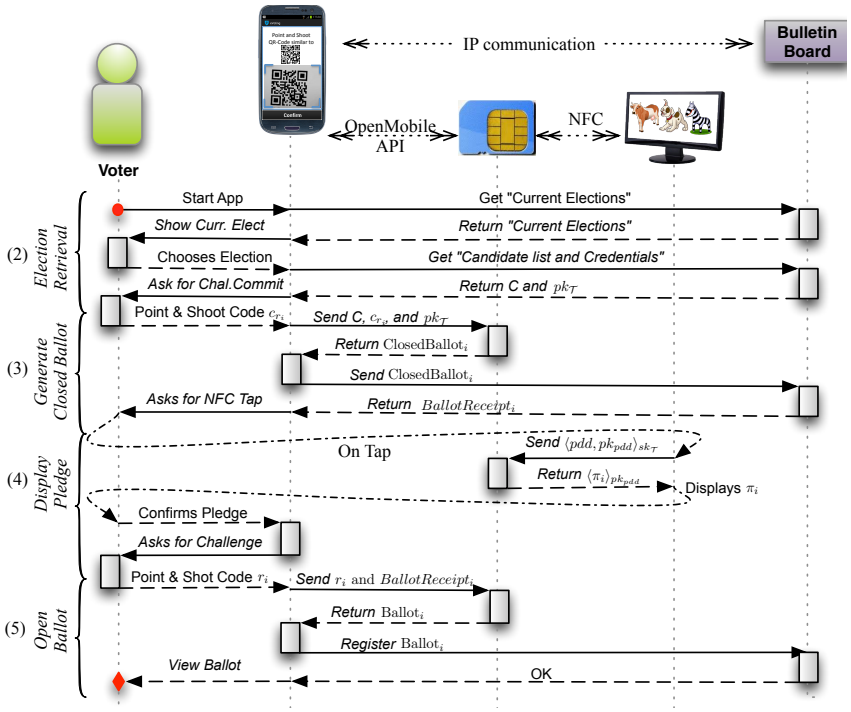


Figure 5.1: Registration procedure.

The registration process starts outside the voting booth with the generation of the 2D-codes and ends inside a private booth. The registration has 5 distinct stages:

(1) 2D-code generation;

(2) Election parameters retrieval;

(3) Closed Ballot Generation;

(4) Pledge Display;

(5) Partial Opening of the Ballot.

The first two stages are just preliminary stages with the intent of gathering information about the election. In the third stage the voting machine (SE) generates and commits to a closed ballot, i.e. a ballot that must be open before being used to vote. In the fourth stage the voting machine displays the "pledge" to the voter, which will be used has a partial key to open the ballot. Finally, in the fifth stage the voter challenges the vote machine with one of the 2D-codes and the voting machine replies with a partial key for the ballot. Only someone with the "pledge" and this partial key is able to use the ballot to vote. Each of these stages is idempotent and starts with the voter and ends with the voter, therefore it may be repeated several times in case of a communication or smartphone failure (e.g. battery depletion). It follows a description of the steps in each of these stages:

1. 2-D Code Generation

   (a) $\mathcal{V}_i$ or $HO \rightarrow \mathcal{V}_i$: QR-code$(r_i), PDF417(c_{r_i})$

The registration starts when each voter $\mathcal{V}_i$ generates, either with the help of an online HO or on her own, one random number $r_i \in_R \mathbb{Z}_q$ and one commitment $c_{r_i} = H(r_i)$, and print the former in a QR-code and the latter in a PDF417 code, which are used in stages (3a) and (5), respectively.

2. Election parameters retrieval

   (a) $\mathcal{BB} \rightarrow$ Smartphone $\rightarrow SE : \langle id_{election}, pk_{\mathcal{T}}, \mathcal{C} \rangle_{sk_{EC}}$

   After generating the two 2D-codes, the voter requests, from the $\mathcal{BB}$, the public key $pk_{\mathcal{T}}$ and the candidate list $\mathcal{C}$ of the election, signed by the electoral commission, and forwards it to the secure element.

3. Closed Ballot Generation

   (a) $PFD417(c_{r_i}) \rightarrow$ Smartphone $\rightarrow SE : c_{r_i}$

   The third stage starts by asking the voter to point her smartphone camera to the PDF417 code and read the commitment $c_{r_i} = H(r_i)$ encoded in it, which is also forwarded to the SE. This step prevents the voter from establishing a covert channel using the random number r_i to leak the "pledge" in stage (5).

   (b) $SE \rightarrow$ Smartphone $\rightarrow \mathcal{BB} : \langle c_{r_i}, ClosedBallot_i \rangle_{sk_i}$

   The SE element replies by generating a closed ballot and forwards it to the $\mathcal{BB}$, through the smartphone. This ballot is similar to an MP3 vote [40] but for a random candidate, i.e. the ballot contains already a full fledged vote but no one knows for which candidate until the ballot is partial open in stage (5). The ballot is comprised by n_c tuples, one for each candidate, with two exponential ElGamal encryptions, each.

$$\begin{aligned} ClosedBallot_i &= \{cv_{ij}\}_{j=1}^{n_c} \\ &= \{\mathcal{E}_{pk_{\mathcal{T}}}(m_{ij}, \tau_{ij}), \mathcal{E}_{pk_{\mathcal{T}}}(\theta_{ij}, \delta_{ij})\}_{j=1}^{n_c} \end{aligned} \tag{5.1}$$

   The first encryption contains $m_{ij} = 1$ for a random candidate $j = p_i \in_R \mathbb{Z}_{n_c}$ (i.e. a YESVote for a random candidate p_i) and $m_{ij} = -1$ for all the others. The second encryption, encrypts a random number $\theta_{ij} \in_R \mathbb{Z}_q$, where $\pi_i = \theta_{ij}|_{j=p_i}$ is dubbed "pledge" of voter's $\mathcal{V}_i$. $\tau_{ij} \in_R \mathbb{Z}_q$ and $\delta_{ij} \in_R \mathbb{Z}_q$ stand for the usual ElGamal secret randomization factors.

   The SE also generates and publishes along with the closed ballot a couple of proofs that $\forall j : m_{ij} \in \{1, -1\}$ and that $\forall i : \exists! j : m_{ij} = 1$, respectively, which are omitted for brevity, and modeled by two oracles $\Omega(v)$ and $\Omega(cv)$, respectively. Check [40, 19, 41] for details.

   (c) $\mathcal{BB} \rightarrow$ Smartphone $\rightarrow SE : BallotReceipt_i = H(\langle c_{r_i}, ClosedBallot_i \rangle_{sk_{\mathcal{BB}}})$

   The $\mathcal{BB}$ replies with a signed receipt for the ballot – $BallotReceipt_i$, which is also shown to the user and optionally printed.

4. Pledge Display

The “pledge” display is the first stage that must and should be done within the controlled precinct. It starts with a tap between the voter’s mobile phone and the PDD. The tap marks the starting of the process that must be done inside a private booth.

(a) $PDD \rightarrow SE : \langle PDD, id_{election}, pk_{pdd} \rangle_{sk_{EC}}$

After detecting the tap the PDD sends its own certificate to the SE inside the mobile phone. The SE checks the signature on the certificate and the election identifier to be sure that the “pledge” will be sent to an official PDD.

(b) $SE \rightarrow PDD \rightarrow \mathcal{V}_i : \langle \pi_i \rangle_{pk_{pdd}}$

The SE takes the public key pk_{pdd} of the PDD’s certificate, encrypts the “pledge” π_i with it, and sends it back to the PDD on the same tap. The PDD decrypts the “pledge”, takes the least α significant bits ($\pi \bmod 2^{\alpha-1}$), converts it to a sequence of symbols and displays them on its screen. When the voter sees the “pledge” on the PDD’s screen she presses the confirmation button on her smartphone

5. Partial Opening of the Ballot

The fifth stage starts by asking the voter to point her smartphone camera to the previously generated QR-Code and proceed as follows:

(a) QR-code$(r_i) \rightarrow$ Smartphone $\rightarrow SE : r_i$

The number read from the QR-code r_i is sent to the SE which checks that it matches the commit-value previously read from the PDF417 code, i.e. $c_{r_i} \stackrel{?}{=} H(r_i)$.

(b) $SE \rightarrow$ Smartphone $\rightarrow \mathcal{BB} : Ballot_i$

If both numbers match, the SE generates a $PartialKey_i$ and builds a complete ballot with it and the previous generated items.

$$Ballot_i = \langle \mathcal{V}_i, ClosedBallot_i, BallotReceipt_i, PartialKey_i, chal_i^n, c_{r_i}, r_i \rangle_{sk_i}$$

The $PartialKey_i$ is comprised by n_c tuples, one for each tuple in $ClosedBallot_i$.

$$PartialKey_i = \{\vartheta_{ij}, \omega_{ij}\}_{j=1}^{n_c}$$

The first element ϑ_{ij} of each of the tuples is either the “pledge” π_i, or some hidden but provable correct value, resulting from the θ_{ij} and some challenge $chal_i^n$.

$$\vartheta_{ij} = \begin{cases} \pi_i & \text{if } j = p_i \qquad (YESvote) \\ 2 \cdot chal_i^n - \theta_{ij} \bmod q & \text{if } j \neq p_i \qquad (NOvote) \end{cases}$$

The challenge $chal_i^n$ must be fresh and unpredictable by the secure element, otherwise the secure element could guess $\vartheta_{ij}|_{j \neq p_i}$ before displaying the “pledge” to the voter and fool the

voter by showing a different ϑ_{ij} as "pledge".

$$chal_i^n = H(r_i, BallotReceipt_i, n) \bmod q, \quad n \in_R \{1, \ldots, \rho\}$$

The unpredictability of the challenge is ensured by the random value r_i encode in the QR-Code under assumption A_{I4}, and the freshness by the $BallotReceipt_i$. For reasons of usability, only α bits of ϑ_{ij} are shown to the user in the next step, therefore it is not unlikely that two or more of these codes would be perceived has equal by the voter. To prevent that, the SE element generates at most ρ different challenges until a $PartialKey_i$ with no perceived duplicates is generated [1].

The second element ω_{ij} of each of the tuples is a proof of correct generation of the first element ϑ_{ij} (check step (1) in section 5.5).

$$\omega_{ij} = \tau_{ij} \cdot (chal_i^n - \vartheta_{ij}) + \delta_{ij} \bmod q$$

(c) Smartphone $\rightarrow \mathcal{V}_i : \{\vartheta_{ij} \bmod 2^{\alpha-1}\}_{j=1}^{n_c}$

Finally, the least α significant bits of each ϑ_{ij} are sent to the smartphone and displayed to the voter encoded in a sequence of symbols. Only then the voter is allowed to leave the booth.

5.4 Voting

The voting phase starts either immediately after the registration ends, using the end screen of the registration, or when the voter runs again the SM&V application on her smartphone and asks the SE for unused ballots (figure 5.2).

1. $\mathcal{V}_i \rightarrow$ Smartphone $\rightarrow SE : rot_i$

   The voter takes the ballot $Ballot_i$ and the official list of candidates and rotates the ballot until the entry of the $YESvote$ matches the chosen candidate. The voter is the only one that knows which entry of the ballot contains the $YESvote$ because it is the only one that knows the "pledge". The chosen rotation is then forward to the secure element.

2. $SE \rightarrow$ Smartphone $\rightarrow \mathcal{BB} : Vote_i$

   The secure element takes the rotation value, signs it together with a digest of the ballot and forwards to the $\mathcal{BB}$,

   $$Vote_i = \langle \mathcal{V}_i, rot_i, H(Ballot_i) \rangle_{sk_i}$$

   which stores it along with the rest of the voter's data

   $$\mathcal{BB}_i = \mathcal{V}_i, Time_i, BallotReceipt_i, Ballot_i, Vote_i$$

   The voter may vote any number of times, only the last one counts. Therefore, if some communica-

tion or smartphone failure occurs in the process, the voter may just repeat the failing step, or the complete voting process.

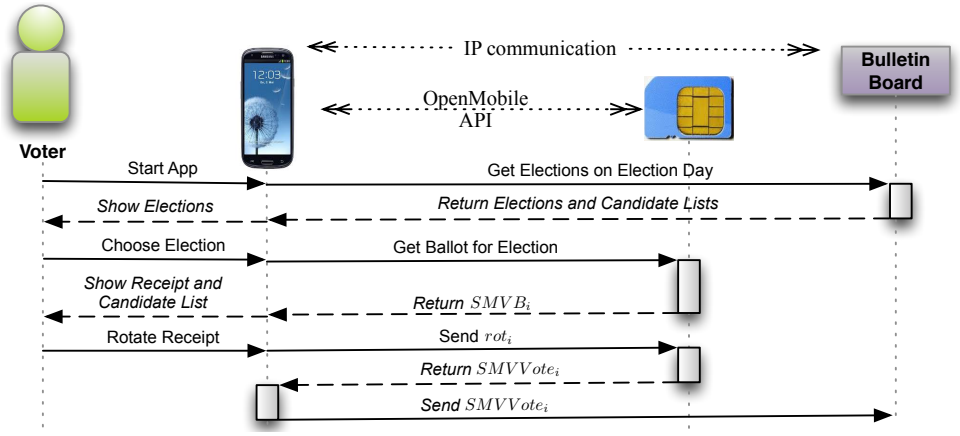


Figure 5.2: Voting message flow.

5.5 Tally & Verification

The tally and verification phase is similar to the EVIV's [1] tally and verification phase. It is comprised by four steps: a preliminary verification step, a deduplication step, a user verification step and a counting step.

1. The preliminary verification step can be carried as soon as each vote arrives to the $\mathcal{BB}$. It consists of verifying:

   i) the signatures of each element in the $\mathcal{BB}$: $BallotReceipt_i$, $Ballot_i$ and $Vote_i$;

   ii) the casted vote is for the registered ballot i.e. the digest $H(Ballot_i)$ within $Vote_i$ matches the digest of ballot $Ballot_i$;

   iii) the challenge is correctly generated i.e. $chal_i^n \stackrel{?}{=} H(r_i, BallotReceipt_i, n) \bmod q, \quad 0 < n \leq \rho$ and $c_{r_i} \stackrel{?}{=} H(r_i)$).

   iv) all encrypted values m_{ij}, within $Ballot_i$, are either 1 or -1, which may be achieved using a Zero-Knowledge Proof such as the one proposed in [40, 19, 41], and is modeled here by Oracle $\Omega(cv)$.

   v) only one of the m_{ij} is 1, which may be achieved using a Zero-Knowledge Proof such as the one proposed in [40], and is modeled here by Oracle $\Omega(v)$.

   vi) the $PartialKey_i$ used to open the ballot was correctly generated by checking the Zero Knowledge proof [40]:

$$\mathcal{E}_{pk_\mathcal{T}}(chal_i^n, \omega_{ij}) \stackrel{?}{=} (\mathcal{E}_{pk_\mathcal{T}}(m_{ij}, \tau_{ij}))^{chal_i^n - \vartheta_{ij}} \cdot \mathcal{E}_{pk_\mathcal{T}}(\theta_{ij}, \delta_{ij}) \tag{5.2}$$

Every non-complying vote is removed from the tally.

2. As the name implies the deduplication step consists on the removal of vote duplicates by checking the time of their arrival. Only the last vote for each voter should remain in the tally. All valid votes and their receipts should be published in the $\mathcal{BB}$.

3. The voter verification step is carried by the voter over the published list of valid votes. The voter is able to check that her vote is cast as intended by checking that the "pledge" in her published vote receipt is positioned by the side of her chosen candidate, that the r_i in the QR-code used to generate the challenge matches the one used at the time of the voting, and that the ballot receipt is the one presented to the voter.

4. Finally, the actual counting is performed by either using a mix net or using the homomorphic properties of exponential ElGamal. Using the homomorphic properties the result of the tally d_j for each candidate j can be computed independently by performing the homomorphic addition of each candidate vote $\mathcal{E}_{pk\mathcal{T}}(m_{ij}, \tau_{ij})$ of each vote $Vote_i$. Given that m_{ij} is either 1 or -1 the homomorphic addition is

$$\bigoplus_{i=1}^{n_v} \mathcal{E}_{pk_\mathcal{T}}(m_{ij}, \tau_{ij}) = \mathcal{E}_{pk_\mathcal{T}}\left(\frac{n + d_j}{2}, \varphi\right) \tag{5.3}$$

   Which is decrypted by the election trustees using their shared secret key $sk_\mathcal{T}$ and solved in order to d_j, which is then publish in the $\mathcal{BB}$ along with a proof of correct decryption (cf. [19]).

Only the last step must be done at the end of the election all the remaining ones may, and should, be done as soon as the vote arrives to the $\mathcal{BB}$.

5.6 Post Election Audit

The Post Election Audit is carried, independently, by several HO and mimics the Tally&Verification phase. The only difference is the actual decryption of the tally result. Instead of feeding the encrypted result to the trustees to be decrypted, HOs compare their encrypted result with the one computed in the Tally&Verification phase, and check the proof of correct decryption provided by the trustees in the Tally&Verification phase.

Chapter 6

Implementation

This chapter describes all the implementation details done in the SM&V prototype.

In the next section the implementation options are described, then in section 6.2 all the SM&V implementation is detailed and finally in section 6.3 some implementation details are specified.

6.1 Implementation Options

This section will describe the implementation options made to achieve the pretended solution. Every decision made will be justified as its impact on the solution.

First of all, SM&V is based on the protocol EVIV which already had an incomplete prototype with most of the components not integrated. In each previous implementation some decisions had already been made that were kept, and other decisions had to be made from scratch.

6.1.1 Web Server and Applications

The Web Application was developed with GlassFish 4.0 using the Interface Development Kit (IDE) NetBeans that is already pre integrated. This decision was made due the good implementation and integration with Java Server Faces (JSF) and Swing for the development of the Web Application and the User Interface Applications.

Most of the Web Server was developed in the EVIV implementation, but new parts needed to be implemented to transform the EVIV implementation to SM&V. Furthermore, previous bugs were fixed and other unimplemented methods were implemented.

6.1.2 Database

The Database decision, like the Web Application, was based on previous EVIV implementation. MySQL Server 5.5 [42] was used. Previous database was adapted for SM&V implementation.

6.1.3 Web Services

All communications between applications and $\mathcal{BB}$ are made through web services using Simple Object Access Protocol (SOAP) messages. The decision to use SOAP instead of other possible solutions was made to reuse, as much as possible, the EVIV implementation. The SM&V implementation uses the ksoap2 Application Programming Interface (API) [43] to create web service requests from the Android application. Some of the already implemented web services were adapted and others were created from scratch to function in SM&V architecture.

6.1.4 Mobile Application

The SM&V implementation of the voting application was done on a smartphone with NFC, running Android 4.0.4, with a GO-Trust secure microSD running Global Platform 2.2.2 [37] and Java 2.2.2 applets.

The Android decision was based on the equipment available to develop and test the SM&V prototype, a Samsung Galaxy S3. The SM&V usability would be severally hindered if more than one NFC messages could not be sent in a single tap. Since the Android NFC implementation do not allow more than one message, Professor Carlos Ribeiro provided two mobile devices with Android version 4.0.4 containing a patched kernel to overcome this limitation.

The Android application was developed using the Android Development Tools (ADT). ADT is a plugin for the Eclipse IDE that is designed to give a powerful integrated environment to build Android applications.

6.1.5 Secure Element

The Secure Element is an essential component in SM&V. It is where all cryptographic functionality is processed in voters device. The system only works if the assumptions defined in section 4.2 are attained.

The voting machine should run in an UICC, avoiding the distribution of another card to the voters, given that the UICC is already available in smartphones and can be provisioned Over-the-Air (OTA). However, running the voting machine in a UICC requires both the collaboration of mobile network operators, which is not always possible, and the availability of the OpenMobile API [44] on the voter's smartphone, which seldom occurs. On the other hand, smartphones with microSD slots are much more common.

Initially, the SE used for the implementation were UICC's cards ordered from the company Gemalto [45] but unfortunately some unexpected problem occurred that forced us to look for another solution. The solution found was a SDCard, a GO-Trust secure microSD card [46] running Global Platform 2.2.2 [37] and Java 2.2.2 applets.

Gemalto Development Tool Kit provides some applications that simplifies the process of development with an Eclipse like IDE, and some applications that help the deployment of applets.

Some of the major problems encountered with the Gemalto solution was related to the official documentation which was inconsistent, some functionalities specified were not supported in the newest

versions and their Development Tool Kit (DTK) only worked on Windows devices.

The main problem that forced us to leave this solution was related to UICC keys. The DTK had already several keys for the variety of cards supported but unfortunately none of them was the right one for the Gemalto SIM card. In Gemalto cards policy, if authentication process fails ten times in a row the card is blocked forever.

The main reason to use the Gemalto cards was related to their NFC capabilities, but with only one card left for the project, this solution was abandoned. After these problem, the NFC communication had to be made through the Android NFC Beam.

The Software Development Kit (SDK) from Go-Trust had some limitation too. It has an application for deploying applets but it is only supported in Windows. There is almost zero documentation which slowed the process.

6.2 SM&V Implementation

This section describes the implementation of the SM&V entities described in section 4.1.

6.2.1 Bulletin Board

The SM&V communication between the voting machine and the $\mathcal{BB}$ is mediated by the smartphone. The $\mathcal{BB}$ API for that communication is the following one:

- setMP3_ClosedBallot – receives three arguments. The voterId, electionId and the actual data sent from voter's smartphone in a byte array.

  First, extracts the Digest of the Closed Ballot, the actual Closed Ballot and the Challenge Commit from the byte array data.

  Then it verifies the signature of the message. To verify the authenticity of the message, $\mathcal{BB}$ gets the intended public key from the database. Since this is stored in Base64, the public key is converted to a byte array and then the public key is recreated. Only after this processes, the signature can the verified, using the SHA1withRSA algorithm.

  After signature verification, if data is not already in the database it is inserted, otherwise the function checks if the maximum registration attempts have been reached. If not, the information is updated in the database.

  The $\mathcal{BB}$ generates the ballot receipt, signs the message and stores the receipt with the digest in the database.

  Finally, the number of registration attempts is merged with ballot receipt and sent back to voter's smartphone.

- getMP3_ClosedBallot – giving the electionId and the voterId, the function gets the closed ballot message from the database;

- setSMV_Ballot – receives the electionId, voterId and the data sent from the voter's smartphone in a byte array. The last argument received is the information of the ballot message (cf. section 5).

  Only part of the ballot comes from the message received because the $\mathcal{BB}$ already has some of the information comprising the SM&V ballot (see more detail in section 6.3.5). Not sending the complete ballot has two main goals, one is to reduce the message length between the smartphone and the $\mathcal{BB}$, and the other, and most relevant, is to reduce the length of the mssage from the voting machine to the smartphone.

  Inside the $\mathcal{BB}$ the information is extracted, namely the digest of the ballot, the partial keys, the ballot challenge and the challenge.

  The actual ballot message, that was composed in SE, is reconstructed and finally the signature is verified. If verification is true, the information is stored in the database, otherwise an error is returned;

- getSMV_Ballot – giving the electionId and the voterId, the function gets the ballot message from the database;

- setSMV_Vote – receives three arguments. The voterId, electionId and the actual data sent from the voter's smartphone in a byte array. It extracts the digest of the vote and the rotation.

  Then it reconstructs the voter message created on the SE so it can verify its signature. If the signature is verified, the information is stored in the database, else it returns an error.

- getSMV_Vote – giving the electionId and the voterId, the function gets the vote message from the database.

In order to have the $\mathcal{BB}$ web services working with the previous functionalities, a Web Service Definition Language (WSDL) and XML Schema Definition (XSD) had to be created and updated.

The database used by the $\mathcal{BB}$ in SM&V is similar to the one used by the $\mathcal{BB}$ in EVIV, however, a few changes had to be made, namely the following tables were changed:

- Election – new columns alpha_bits, max_challenges and max_registrations were added to the table. Each of these variables are defined as integer variables with a maximum of 11 digits;

- Votes – new columns registration_attempt, challenge_attempt, challenge_commit, challenge, MP3_ClosedBallot, MP3_BallotReceipt, SMV_Ballot and SMV_Vote were added to the table to achieve the intended architecture. The first two are integer variables with maximum of 11 digits, and the rest are Binary Large Object (BLOB) variables.

6.2.2 Electoral Commission

One of the roles of the Electoral Commission is the voter enrollment, which is performed with the help of a specific application that configures each SE before it is handed to the voters. The application performs the following steps on each SE:

- Retrieve voters public key – sends a request to the SE through an Application Packet Data Unit (APDU). The ECApp reads the response data and reconstructs the public key, which is then converted to base64, stored in the $\mathcal{BB}$ database and displayed to the user (cf. figure 6.1).
- Set Voter Id in the applet – send an APDU with the Voter Id;
- Send EC Certificate – first, the ECApp reads the EC certificate and gets the public modulus and exponent. Then sends it through an APDU so that the SE can reconstruct the public key;
- Send $\mathcal{BB}$ Certificate – first, the ECApp reads $\mathcal{BB}$ certificate and gets the public modulus and exponent. Then send it through an APDU so that the SE can reconstruct the public key.

These operations are performed in sequence when pressed the “Get Public Key from SE” button presented in figure 6.1.

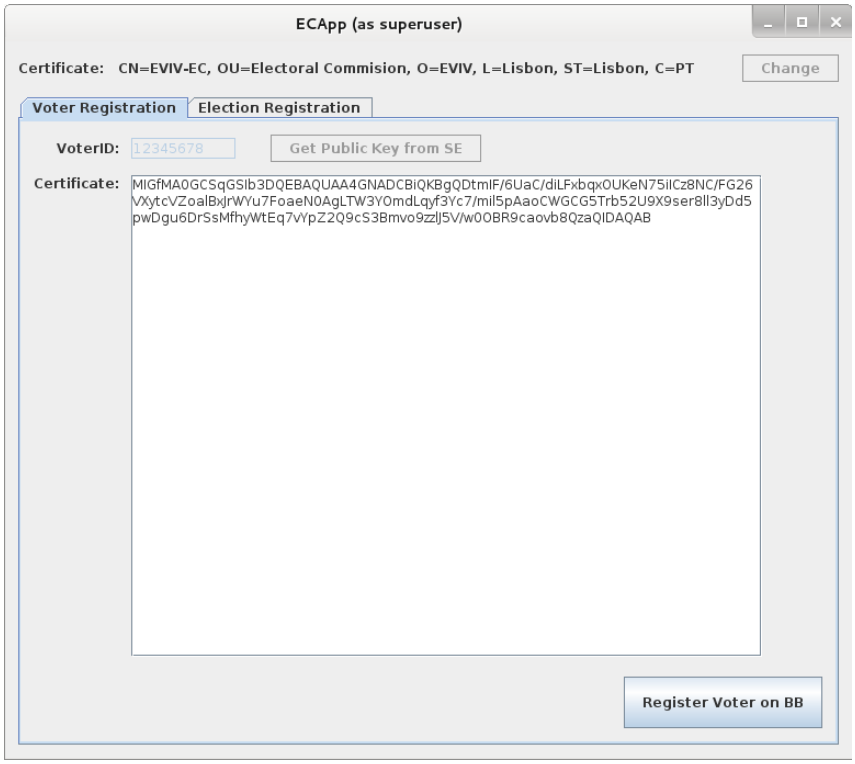


Figure 6.1: Retrieve voters public key, Set voter’s id, send EC and $\mathcal{BB}$ certificate.

The outcome of every function that sets/gets information to/from SE is stored in a file that is processed in the Electoral Commission Application (ECApp). For example, to retrieve the voter’s public key, SE first composes a message that contains the public modulus and the public exponential, passes through the C program which stores it in a file, and at the end, the ECApp reads this information and reconstructs the public key.

The following options were added for the election configuration tab of the ECApp:

- Max registration attempts;
- Max challenge creation attempts;

- Alpha Bits (least significant bits used to generate verification codes)

Another task that the EC must perform is the credentiation of the PDD's to a specific election.

Each election may have several PDD's, and for each one, $\mathcal{BB}$ must associate its identification, election participating and its certificate. There are some implementation rules defined that are specified next.

Since non-repudiation property is not necessary in PDD keys the creation and PDD configuration is done in the EC, the PDD keys are generated by the EC and inserted into the PDD afterwards.

In an attempt to implement PDD's as generic as possible, when EC assigns a PDD Id to the pretended election (button "Assign PDD Id" in figure 6.2) it generates a pdd.ini file that contains PDD Id, Election ID and Alpha Bits that are election parameters for PDD.

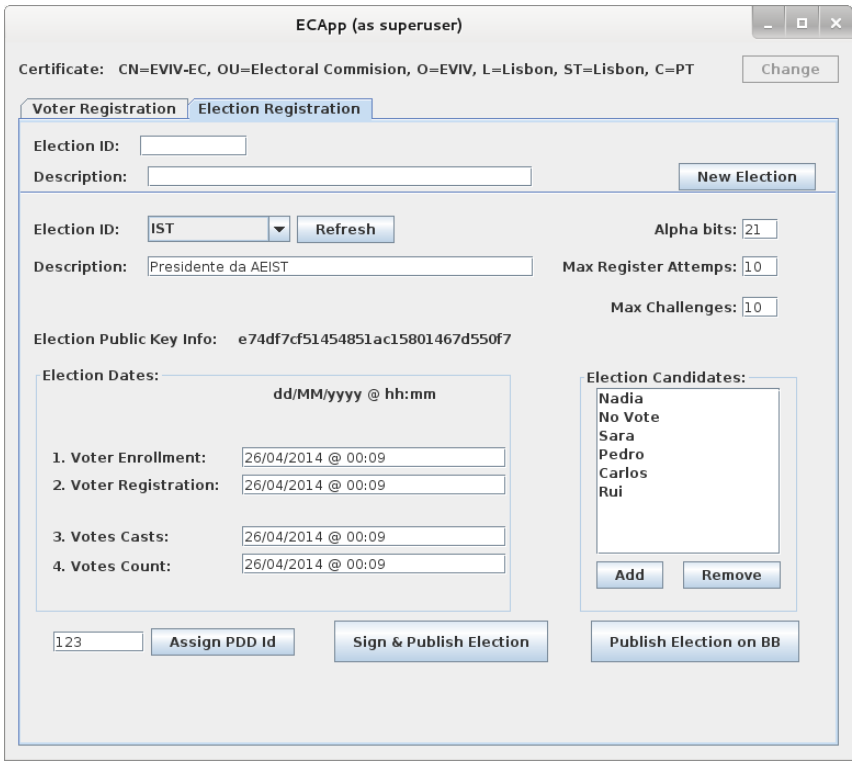


Figure 6.2: Setting the election parameters.

The first message that each PDD sends to the voter at registration is generated and signed by the EC. This message is generated with the generation of the PDD's id by the EC and stored in a specific file: "pdd_id.msg". The generated message contains the PDD's certificate, i.e. a signed statement of the PDD's id and the Public Key.

6.2.3 Android

The Android application has four main functions, corresponding to the four choices of the main menu: Register, Vote, Verify and Exit.

The Registration function is the most complex one. It may be described along the screens seen by the voter upon choosing this function (cf. figure 5.1).

The first screen (screen a) is the main menu containing the above four functions. The remaining screens are described below.

Select Election (screen *b*)

Figure 4.2 screen *b*, presents a list of current available elections to the voter. The election list is presented to the user only after the information is received through a thread running a web service that retrieves the information using the getElectionList function. If the information is retrieved successfully it presents the election identification and its description. If any error occurs, the application finishes the current activity and shows an error.

Next, the voter selects the election she pretends to participate and a web service is called , getElectionInfo(), to get the actual election information. The information comes in a SOAP message. To parse this information and store it in the smartphone, a personalized parser was created. The parser checks the message for every tag that the election information can give, namely: electionCandidates, electionID, electionDesc, electionDates. alphaBits, maxChallenges, electionPubKey. Then it stores them in a static class so the information can be accessed in any part of the registration.

Scan PDF417 (screen *c*)

The next activity, screen *c*, asks the user to point and shoot to a PDF417 after clicking the button. One of the challenges encountered was to find an application that could read PDF417 and QR-Code for free. This was done with the PDF417 Barcode Scan API [47]. It was the only solution possible for the intended purpose. The only alternative was ZXing, but it could only read alpha quality PDF417 and in 50 attempts, it could only read one. Fortunately the PDF417 Barcode Scan could scan both PDF417, as the name says, and QR-code very fast compared to the ZXing. Initially some problems occurred with the solution picked but the company was very supportive and helped me a lot and fortunately I could help them too with a unknown bug they had in their API and with the correspondent solution.

After scanning the PDF417 the smartphone begins the communication with the SE. These communications are made through APDU messages. In an attempt to maintain the APDU organized some rules were made (for details see –section 6.3.2). In this step the following APDU messages are sent to the SE:

- Set Max Challenges – send the maximum number of challenges that can be generated;
- Set Max Symbols Combination Bits – gets the number of possible symbols in that specific election and sends the length of bits to the SE. The number of bits is sent because of the way some functions were implemented in the SE as is described in section 6.2.4;
- Set Alpha Bits – send the number of bits that the value of the pledge is truncated to;
- Set Challenge Commit – send the challenge commit scanned with the PDF417 Barcode Scan application;

- Set Election Parameters P, Q, G, H, MP_G and MP_G_INV – this information comes from the ElGamal public key generated in the Trustee application, it was retrieved in screen *b*;
- Prepare Ballot – prepares the ballot for each candidate. To do so, the P1 header from the APDU message is changed until each candidate index in the SE is sent;
- Create Candidate Encryption – sends an APDU for the SE to create the candidate encryptions;
- Create CGS97 Candidate Proofs – sends the APDU that triggers the creation of the CGS97 candidate proofs;
- Create Closed Ballot – send the APDU to the SE to create the closed ballot message;
- Get Closed Ballot – receives the closed ballot that is composed by the challenge commit and closed ballot signature and the actual data of the closed ballot. Due to the message size it needs more than one APDU to receive all the information (more detail in section 6.3.2).

  Before sending the information to the $\mathcal{BB}$ the challenge commit has to be merged with the message received from the SE. The SE does not send the challenge commit, because the smartphone already has that information (more details in section 6.3.5);

  After merging the data, the web service setMP3_ClosedBallot() is used, which returns the ballot receipt and the registration attempts already done from the voter;
- Send Ballot Receipt – send the ballot receipt received from the $\mathcal{BB}$ to the SE;
- Send Registration Attempts – send the registration attempts made from the voter to the SE;
- Verify Ballot Receipt – send the operation to verify inside the SE if the receipt matches the signature of the closed ballot.

Since all these operations can take some time to perform (see table 7.4) the SM&V application presents a loading screen that calculates the predicted time depending on the number of candidates and shows the progression to the voter.

Tap Pledge Display Device (screen *e*)

In this screen, the voter's smartphone when tapped with the Pledge Display Device (PDD) communicates through Near Field Communication (NFC). As mentioned previously in section 6.1.4 the android kernel was patched to allow more than one message to be exchanged in only one tap.

The implementation code to make this happen was already done from a previous thesis, and it was just adapted to function in the SM&V mobile application.

In a general view the Android SM&V application and the PDD interaction can be seen in figure 6.3. The voter application in this step waits (Wait) for the PDD to start the communication (Beam).

After the application receives the beam message in the wait activity, it sends the next APDU messages to the SE:

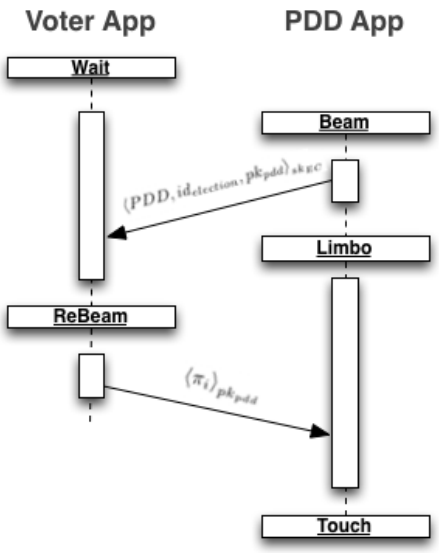


Figure 6.3: NFC messages exchange between the Voter App and the PDD App.

- Set PDD Message – send the actual PDD message to the SE. This message contains the voter identification, the PDD identification and the PDD certificate;
- Set Signed PDD Message – this message is generated in the EC, it is the signature of the above message;
- Get the Encrypted Pledge – this message is only used in the next activity (ReBeam) but it is performed in this current activity to maintain the APDU messages in the same place.

Then it starts the ReBeam activity to send the encrypted pledge retrieved in the Wait activity.

Pledge Display Device Application

It was defined that every PDD in its preparation phase for the election receives its own certificate and pdd.ini, pdd.msg files from the EC.

The first time that PDD starts, loads its own certificate and stores other information from the pdd.ini file (its own identification, the election that it is going to participate in and the alpha bits pre-configured value for the election).

The pdd.msg, as mentioned in section 6.2.2, contains the message that the PDD is going to exchange with the voters and is composed by the PDD identification, the election that it is going to participate in, its own certificate and all the previous information signed by the EC.

This next step is optional, but to guarantee that the pdd.msg was authentic, the PDD reconstructs the information and verifies its EC signature, and if no problems occur, it starts the Beam activity to interact with the voters on tap (figure 6.3).

In the first interaction with the voter application, the PDD message signed by the EC is sent. In the Limbo activity it receives the response. This response contains the encrypted pledge that the PDD is

going to display to the voter. This information is transported to the final activity, Touch, and it starts the process to present the pledge.

In order to decrypt the PDD message, it needs to use the same algorithm used to encrypt the pledge in the SE, RSA with the PKCS1 padding algorithm (more details in section 6.2.4). Since the Cipher class in Java does not provide these algorithm combinations, an external provider was used to make this possible named Spongy Castle [48] that is a fork of the known API Bouncy Castle targeted to the Android environment.

After decrypting the encrypted pledge, the PDD only needs to present it in the screen, so it gets the number of each symbol that needs to be presented to create the pledge. This process execution is explained in detail in section 6.3.6.

Finally a sequence of three symbols from the list of all symbols already in the PDD application is created.

Scan QR-Code (screen *g*)

The next screen in the registration phase is to get the challenge generated with the help of the Helper Organization (HO) through a QR-Code.

This step is similar to screen *c* since it uses the PDF417 Barcode Scan to read the QR-Code.

After reading the challenge, the following APDU messages are sent to the SE:

- Set Challenge – send the challenge read in the QR-Code to the SE;
- Verify Challenge – after setting the challenge, the SE needs to verify if the challenge commit is really the hash of the challenge. If it corresponds, the execution can continue;
- Create Verification Codes Symbols – requests the combination of symbols for each verification codes from the SE;
- Get Ballot Length – since the ballot information can vary depending on the number of candidates, first an APDU to know the length of the incoming next data is sent;
- Get Ballot – gets the actual ballot message. If the ballot message is greater than one APDU, then more than one APDU is sent to retrieve the various chunks of the information (more details in section 6.3.2). After reconstructing the ballot message, the challenge commit is merged to the previous messages.

After executing the previous operation, the ballot message needs to be stored in the $\mathcal{BB}$. To do so, the web service setSMV_Ballot() is executed. If the web service returns true, it can proceed to the next step.

Since some of these steps can be time consuming, as it can be seen in table 7.4, the application presents a loading screen while the SE is processing the requested functions.

Presenting the Ballot (screen *h*)

The final screen in the registration phase is the presentation of the ballot.

In this screen, two lists side by side are presented on the device. One of the lists contains all the election candidates and the other vote codes. One of these vote codes is correct and the others are false. The voter votes by scrolling one or both lists and matches her chosen candidate with the correct code.

To create the sensation to the voter that both scrolling lists can be infinitely scrolled, the set of elements (candidates and verification codes) were repeated at least two times and at most the times necessary to occupy more than one screen.

Contrary to most applications that improve usability with a smooth scroll, this screen is better implemented with a grid-like kind of scroll that ensures that candidates and verification codes are always perfect aligned. Figure 6.4 shows on the left side the result of the current implementation, and on the right side the implementation that would result with a smooth scroll.

Another implementation choice was the connection between the scrolling of the two lists. The list with the verification codes scrolls freely at the will of the voter. However, when the voter scrolls the candidate list, both lists scroll together. This feature allows the voter to easily check the verification code assign to every candidate and use one of them as a false pledge if she is coerced.

Figure 6.4: In the left the correct example on the way to match both lists. On the right the wrong way.

To present the actual set of verification codes for each candidate, the application converts the unique combination of symbols that were transported from the previous step to the actual index number of each symbol (more details in section 6.3.6). Afterwards it creates an image composed by three sets of symbols for each candidate.

Voting

The voting phase is very similar to the last screen *h* of the registration.

To vote, the smartphone needs to store the rotation done from the pledge list relative to the candidate list. For example, if only the candidate list is scrolled and since in that situation both lists are scrolled synchronously, the rotation is zero, but if only the pledge list is scrolled down one position, the rotation variable is set to one.

The voter commits with her choice by clicking the vote button, upon which the following commands are sent to the SE:

- Set Rotation – send the rotation made to the SE;
- Create Vote Message – request to create the vote message inside the SE;
- Get Vote Length – get the length of the vote message because it can vary depending on the number of candidates;
- Get Vote – gets the actual vote message composed by the SE signature of the vote message, the voter identification, the rotation and the hash of the ballot to guarantee the SE is using the same ballot.

After the SE finishes the operations, the smartphone calls a web service named setSMV_Vote() which stores the vote message in the $\mathcal{BB}$ database.

Verify Casted Vote

The verification menu has two options: the verification of the vote and the election tally.

To verify her own vote, the application asks the user to select her challenge QR-Code used in the election, and after that it displays the Vote with the pledge already matched.

The difference in the verification process is that, it is impossible to change the match between the candidate and the pledge.

Unfortunately the tally process was not merged in time with the Android application, but supposedly there are already some implementations from the previous thesis that could be used for the tally process.

6.2.4 Voting Machine

A lot of functionality was already implemented from the previous Ph.D thesis by Rui Joaquim, since SM&V is based on the EVIV implementation.

In this section only the new implementation is described. The communication between the smartphone and the voting machine, within the SE, carried by APDU messages.

Applet Deployment

In the voter enrollment phase the EC gives an SDCard to the user that contains an applet. The GO-Trust SDK is used to deploy the applet developed (the voting machine). But first the applet needs to be compiled in a *cap* file (more details in section 6.3.1).

Since the SDCard has some space limitations, in the deployment execution, all variables relevant to the SM&V protocol are stored in persistent local variables with predefined size.

Some of the variables are used frequently (e.g. variable that handles the response data). In those situations and if their maximum size is less than 1 KByte, they are defined as transient. These variables are stored in the Random Access Memory (RAM) instead of Electrically Erasable Programmable Read-Only Memory (EEPROM). Their reading and writing speed is much faster than persistent variables, but they are cleared when the SE is not powered.

On other variables it is impossible to know ahead their size because they can vary the size according to the number of candidates. The work around to this problem is described in section 6.3.4.

When the applet is deployed it initializes the previous variables and the voter electing credentials are generated.

After deploying the applet, the SDCard is provisioned by the ECApp with the following actions:

- Voter Identification – set the voter identification;
- Get Voter Public Key – get the voter public key generated in the deployment time. For this operation, since it is only possible to send byte arrays, a variable was created that contained the public exponent and modulus, to be reconstructed afterwards on the destination device;
- $\mathcal{BB}$ Public Key – set the $\mathcal{BB}$ public key. Like the previous public key, the message comes with the public exponent and modulus information. To reconstruct the public key a RSAPublicKey instance is created and then the exponent and modulus are assigned;
- EC Public Key – set the EC public key. The process is the same as the previous operation but for the EC public key.

Configure Election Variables

Depending on the election selected by the voter, some configuration variables need to be set. It is important to mention that every function is organized in a way that every “set” APDU has an equivalent “get” APDU that differs only on the first byte – the P1 byte, more details in section 6.3.2.

- Election Identification – set and get the election identification;
- Registration Attempts – set and get the registration attempts already made in that particular election;
- Set Alpha Bits – set the alpha bits predefined value;
- Set Max Challenges – set the maximum number of times a challenge can be generated.

SM&V Functionality

Besides the APDUs used to configure the election variables, the SM&V vote machine requires the implementation of a number APDU messages to implement the actual voting protocol. Below a list of such APDU is presented (more details in section 5):

- Challenge Commit – set challenge commit value scanned from the PDF417 in the voter's smartphone;

- Create Closed Ballot – in this function, the message closed ballot is created. To do so, in a first step all the ballot encryptions and candidate encryptions from each candidate are merged.

  Then, a digest message is created composed by the challenge commit and the previous merged data.

  Afterwards, the digest message, the challenge commit and the merged data is signed with the voter's private key and finally it creates a final message intended to be sent to the destination device composed by the $\langle ChallengeCommit + ClosedBallot \rangle_{sk_i} + ClosedBallot$. In this previous message, the challenge commit is not sent because the value is already in the smartphone.

- Get Closed Ballot – sends the closed ballot message to the smartphone. Since it does not fit in only one APDU, a solution had to be found to send all the information in different APDU that is described in section 6.3.2;

- Set Ballot Receipt – set the ballot receipt message. This receipt is the response message from the $\mathcal{BB}$ to the closed ballot and is composed by the $Hash(\langle ChallengeCommit + ClosedBallot + RegistrationAttempts \rangle_{sk_{\mathcal{BB}}}) + RegistrationAttempts$;

- Verify Ballot Receipt Signature – after receiving the ballot receipt, the SE verifies its signature. Since the $\mathcal{BB}$ public key was already set from the deployment phase, the SE only needs to reconstruct the message merging the $ChallengeCommit + ClosedBallot + RegistrationAttempts$, and verifying its signature. If the message corresponds, the SM&V process can proceed;

- Set PDD Message – this APDU, depending on the P1 value, does two different operations. When P1 = 0, the SE stores the PDD message in a temporary variable and the digest in another. When P1 = 1, the SE extracts the PDD message stored in the temporary variables and reconstructs the PDD public key. After this process, the SE verifies its signature;

- Encrypted Pledge – since the SE already has the PDD public key from the previous function, this operation is intended to encrypt the voter's pledge with the PDD public key. To do so, the pledge is ciphered with the RSA algorithm with PKCS1.

  Since the PKCS1 padding algorithm is used, it is only possible to encrypt the key size length minus eleven bytes reserved for padding. So, since the key size is the same size as the pledge, it was necessary to separate the pledge encryption in two different APDU messages.

- Challenge – set challenge value. It is important to check if the $Hash(Challenge)$ is equal to the Challenge Commit. To do so, a MessageDigest instance is used, with the same algorithm used to hash the Challenge Commit, SHA 256;

- Create Verification Codes – this function is composed by the next steps:

First the ballot challenge message $Hash(challenge + BallotReceipt_i + ChallengeCount)$ is created. To do so, the previous messages are merged and then hashed. This Hash is then used to create the vote receipt. The receipt was already implemented from EVIV, the difference being that SM&V also creates the verification codes for each and every candidate.

Then, for each candidate, an unique combination set of symbols for each verification code is necessary. The next operations specify the flow execution to achieve this goal:

- After calculating the number of possible verification combinations, the least significant bits are truncated in the verification code. To do so, an arithmetic modulus operation for byte arrays was created, since JavaCard only has a modular operation for short variables.
- Each time a set of symbol is calculated, there is a verification process to check if it is unique. If it is not unique, a new ballot challenge message needs to be created, incrementing the Challenge Count variable, and the vote receipt is recreated.
- If the maximum Challenge Attempts reaches the limit, the process fails.

If this process succeeds, the number that represents the combination set of symbols is then transmitted to the voter's smartphone.

- Get Ballot Message – this APDU is intended to create and respond the Ballot message.

  First the Partial Keys message is created. To perform this operation, the verification codes and their encryption factor are merged.

  Then, the complete Ballot message is merged to afterwards be signed by the voter. This Ballot message is composed by the Voter identification, Closed Ballot, Ballot Receipt, Partial Keys, Ballot Challenge, Challenge Commit and the Challenge.

  Since the $\mathcal{BB}$ has already some of this information on its database, the signature of the Ballot message, the Partial Keys message and the Ballot Challenge are the only information sent back to the voter's smartphone (more details in 6.3.5).

- Rotation – in this APDU, if the P1 value is zero, then it sets the rotation value. If it is one, it returns the rotation;

- Get Vote Message – this APDU is separated in three different functionalities.

  If P1 equals zero, it creates the Vote message. The voter identification, the rotation and the signature of the ballot is signed with the voters private key and is then stored in a local variable.

  If P1 equals one, the SE sends the length of the Vote message

  If P1 equals two, it sends the actual Vote message.

6.2.5 Helper Organization

The Helper Organization has an application to help the registration process. This Application generates one random number and one commitment, and prints the former in a QR-code and the latter in a PDF417

code, which will be used in screen *(c)* and *(g)*, respectively from the figure 4.2.

Figure 6.5: Generate the Challenge and the Challenge Commit.

The QR-code and PDF417 are generated with the ZXing API [49]. A 128 random alphanumeric value corresponding to the QR-code is created. It had to be alphanumeric since the ZXing only could create the pretended 2D-Codes with UTF-8 string values. Next, the QR-Code is hashed with SHA-256 algorithm which generates the PDF417.

Each image is stored locally and presented in the application after pressing the “Generate challenge” button. To generate different sets of 2D codes, the images have to be flushed after generation.

6.3 Implementation Details

With the objective to maintain the previous section as readable as possible, this section is meant to clarify and explain some implementation details as well as the decisions made to create a concise and coherent implementation.

6.3.1 Applet Generation

Two ways were found to generate the applet. Eclipse has a plugin to use JavaCard that has the function to generate a cap file. But in an attempt to automatize the execution process there was another way to generate the applet file. So I used a terminal line command to generate the applet file. The command syntax is described below, enumerating the used arguments:

1. ./converter – the actual command to execute;
2. -classdir /home/goncalo/workspace/MP3Applet/bin – sets the root directory where the converter will look for classes;

3. -exportpath /home/goncalo/Downloads/java_card_kit-2_2_2/api_export_files – specifies the root directories in which the converter will look for export files;
4. -applet 0x01:0x02:0x03:0x04:0x05:0x06:0x07 mp3.Mp3 – sets the default applet Application Identifier (AID) and the name of the class that defines the applet;
5. -d /home/goncalo/Downloads/mp3 – sets the root directory for output;
6. -v – enables the verbose output;
7. mp3 – the applet name;
8. 0x01:0x02:0x03:0x04:0x05:0x06 – the class identifier;
9. 1.0 – applet version.

6.3.2 APDU Organization

The messages to communicate between Android and the SE are called APDU. In summary an APDU message contains a mandatory 4-byte header (CLA, INS, P1, P2) and from 0 to 255 bytes of data, if the APDU is intended to send information, it is necessary to fill the first byte with the length of the data, and if it is expecting output specify the length of the data that is going to be received.

- CLA, 1 byte, Instruction class - indicates the type of command, e.g. interindustry or proprietary
- INS, 1 byte, Instruction code - indicates the specific command
- P1-P2, 2 bytes, Instruction parameters for the command, e.g. offset into file at which to write the data

GO-Trust secure microSD only allows the use of CLA that leaves the two right most bits at zero. Unfortunately no information was provided by GO-Trust that explained why it had that restriction. So for example only CLA 0x00, 0x04, 0x08, etc would work correctly, the other cases would get an error.

The P1 byte is separated in 3 options:

- 0 - Execute pretended function
- 1 - Return the length in bytes of the response data
- 2 - Return the actual data

If the data intended to send is greater than 256 bytes and consequentially does not fit an APDU message, it is separated in several chunks and sent one at a time changing the P2 parameter, so for example, if the length of the message is 700 bytes (3 chunks) the smartphone will ask the same APDU only changing the P2 byte for the pretended chunk and after receiving all chunks, it merges the information.

The JavaCard 2.2.2 supports Extended APDU's that makes possible with a single APDU send more than 256 bytes, but unfortunately, the GO-Trust SDCard does not support this functionality.

6.3.3 Data Structure

In an effort to implement the best programming practices, when two different byte arrays are merged, before each content, two bytes are inserted. These bytes are intended to specify the next data length.

Initially only one byte was used, but in some exceptional cases the length could not fit in only one byte. To maintain the content coherent in every part of the implementation the above solution was adopted.

This decision make the implementation more flexible because there are some information that can vary their size based on the initial predefined values (e.g. key sizes).

6.3.4 Memory Issues

The SE has some limitation that determined some implementations decisions.

One of them is related to its capacity. The used SE has 144 KBytes EEPROM and 4 KBytes RAM. Related to the capacity, it is worth to mention that JavaCard 2.2.2 does not support garbage collection (JavaCard 3.0.2 supports the functionality).

The SE development had to be slightly different from the normal. Every time a new variable was instantiated, memory was allocated in the card, and there is no way to reused it or free that memory (there is a way, but GO-Trust SDCard do not support it).

Based on these memory problems, almost all the variables, specially the ones that occupy more memory where instantiated on deploy time of the applet. Unfortunately some of the variables are impossible to know in advance their size (they depend on the number of candidates). The work around solution to this problem, is to initialize the variables with a size already prepared for the maximum number of candidates in SM&V (10 candidates, already defined from the EVIV protocol [1]).

The only way possible to make more than one full execution in SM&V was reusing the same variables. The variables that needs to be record separately has their own local variable, the rest use large buffers to achieve their goal. The large buffer data used is composed by the largest data that is possible going to be used in SM&V process:

$$Ballot_i = \langle \mathcal{V}_i, ClosedBallot_i, BallotReceipt_i, PartialKey_i, chal_i^n, c_{r_i}, r_i \rangle_{sk_i}$$

- $DigestBallot_i = VoterKeySize\ bytes$;
- $\mathcal{V}_i = 8\ bytes$;
- $ClosedBallot_i = MaxCandidates * 4 * pSize\ bytes$;
- $BallotReceipt_i = BBKeySize\ bytes$;
- $PartialKey_i = MaxCandidates * 2 * qSize\ bytes$;
- $chal_i^n = qSize\ bytes$;
- $c_{r_i} = MD5Size\ bytes$;

- $r_i = qSize\ bytes$

Currently, the implementation can only handle one election at a time due to the SE memory limitation.

6.3.5 Optimizations

There are some optimizations made related to the information sent from the SE to smartphone and smartphone to $\mathcal{BB}$ and vice versa. In some cases there was no need to send all the information from one point to another.

Giving an example, the Ballot message information sent from the SE to the $\mathcal{BB}$, is the only unknown information. So in the Ballot message the SE only sends the DigestBallot, ParitalKey and Challenge to the smartphone. The smartphone merges to this previous message the Challenge and sends it to the $\mathcal{BB}$. At the end, the $\mathcal{BB}$ recreates the complete Ballot message and verify the signature. If the server could successfully verify the signature, the process can continue, so in this specific case, the Ballot message is stored in the database.

When joining the information to recreate the message it is important not to forget the two bytes header before each content referred in section 6.3.3.

6.3.6 Verification Codes Symbols

An algorithm to display the verification codes symbols was implemented in the smartphone and SE. In the smartphone it is needed so that the voter can vote and to verify its vote. In the SE was implemented because it is important to verify if the set of symbols in the ballot are unique.

The implementation in the SE is a little bit more complex than the smartphone because there were some mathematical functions that needed to be created. Next is a brief description of the execution process:

1. Calculate the maximum number of possibilities $(2^7)^3$;
2. For an easier memorization, only the least significant α bits were taken, so it was used the modulus operation. This operation was not implemented yet for a set of byte arrays, so it was implemented.
3. Apply the modulus operation in the previous result with the maximum number of possibilities. These results are the combination set of symbols.
4. There is a process to verify in each execution if the combination was already in the list, because different verification codes could get same verification code symbols combinations. So if that happens, the process is re-done a to a maximum of ρ attempts until all the verification codes symbols combinations are different.
5. After sending the combination to the smartphone, transforming the combination number to the set of symbols it basically converting the number in a function like $a \cdot (2^7)^2 + b \cdot (2^7) + c$ in which the a, b and c represent respectively the first, second and third symbol.

Chapter 7

Evaluation

To validate the SM&V system, an evaluation was conducted. This chapter explains the usability options and their impact, the performances achieved with the SM&V prototype and finally a general discussion of security of the protocol is done.

7.1 Usability

Usability is a major issue in any voting system, but assumes a specific relevance in end-to-end voting systems, where the voter distrusts her voting machine and is, therefore, required to handle a complex voting interface.

SM&V requires the voter to be able to memorize the "pledge" for a long period (sometimes over a month) and be able to distinguish it from the remaining of the verification codes. Both the "pledge" π_i and the verification codes ϑ_{ij} are large random numbers in $\mathbb{Z}_q$, not suitable for memorization or display to the voter. To be displayable, these numbers are truncated to α bits, by applying the $\mod 2^\alpha$ operation (see section 5.3), and coded into a sequence of different symbols. The size of the sequence of symbols depends both on the value of α and on the number of different symbols. A small sequence of symbols simplifies memorization, but implies a larger set of symbols, which complicates distinguishability, therefore, it is expected that choosing the correct set of symbols may have a significant impact on the overall usability of the system.

According with Bertin [50] there are 8 visual variables that are used by humans to distinguish symbols: shape, size, color, brightness, pattern, orientation and horizontal and vertical position. Symbols that differ in more variables are easer distinguishable from each other; therefore it is possible to use large sets of symbols provided that they differ in as many as these variables as possible. On the other hand, long term memory of humans works better with semantic information [51] rather than abstract information, which seems to point that symbols representing concrete concepts are preferred over abstract ones.

The proposed system uses 128 different symbols, varying both in color and in shape, representing

Age	Gender			
	Male		Female	
15-24	5	(11.4%)	8	(18.2%)
25-49	22	(50%)	6	(13.6%)
50-64	2	(4.5%)	1	(2.3%)
> 64	0	(0%)	0	(0%)

Table 7.1: Distribution of subjects by age and gender

Education Level	Percentage
Basic Education	18.5%
Secondary Education	14.8%
College Education	66.7%

Table 7.2: Distribution of subjects by education level

Memo technique	Number	
Sequence of symbols of the "pledge"	15	(29.4%)
Non repeating symbol of the "pledge"	12	(23.5%)
Candidate in front of the "pledge"	8	(15.6%)
"Pledge" position within the ballot	7	(13.7%)
History with the symbols of the "pledge"	3	(5.88%)
Other	6	(11.8%)

Table 7.3: Memorization techniques reported by the voters

128 different objects and animals[1]. Both the "pledge" and the verifications codes are shown as combinations of three of these symbols, therefore $\alpha = 21$, which for a 10 candidate election ($n_c = 10$), with $\rho = 3$, the probability that the secure element is able to trick a voter is $p_{fail} = 1 - p_{soundness} = 1 - (1 - 2^{-\alpha})^{\rho.(n_c-1)} < 10^{-4}$ and the probability of being generated one invalid ballot is no more than $p_{inv} = e^{-\rho.n_c(n_c-1)/2^{\alpha+1}} < 10^{-4}$ [1], resulting, therefore, a highly sound election.

The quality of the chosen set of symbols was tested by an experiment with 45 different subjects, with the distribution of age, gender and education level according with Tables 7.1 and 7.2. To each of the subjects it was shown a sequence of three symbols similar to the "pledge" and a list of sequences of three symbols similar to the ballot. It was then asked the subjects to find the "pledge" in the ballot and memorize both the "pledge" and the position where it appears in the ballot. A copy of the ballot was given to the subjects, who were also instructed not to make any mark or written annotation about the "pledge". Finally, a month later, the subjects were asked to point the "pledge" in the ballot.

The results are very promising, although there is still some margin for improvement. Only 3 of the 45 subjects (6.7%) were not able to point the "pledge" within the ballot, resulting into $93.3\% \pm 6\%$ correctness for a confidence level of 0.9. The reasons for these errors were completely transversal to gender, age or education level. From the three subjects that forgot the pledge, two made a confusion about two of the symbols that were too much alike, and the other mistakenly identified a code similar to the "pledge" of a previous experiment. These two types of mistakes confirmed the relevance of a good choice of symbols (they should be very different from each other), and revealed that consecutive elections should not share the same set of symbols. Both problems may be solved easily.

Nevertheless, voters that forget the "pledge" or are uncertain of it, may register again and receive another "pledge", or may even decide to invalidate their Internet registration and vote using the classical way or any other voting method.

The voters that chose the correct "pledge" reported several techniques to memorize it (Table 7.3).

[1] Taken from the popular game *Categories*.

Registration Screens (Figure 4.2)	Delay between screens ($n_c = 10$)	Registration steps (Section 5.3)	Actions performed between screens	Time (s)
b) $\rightarrow$ c)	0.2 s	(2)	Election parameters retrieval	0.2
c) $\rightarrow$ d)	4.2 min	(3a)	Set challenge commit	0.04
		(3b)	Create closed ballot	$8n_c + 0.9$
			Create ZK proofs	$16.2n_c$
			Get closed ballot from SDCard	$0.4n_c$
			Get ZK proofs from SDCard	$0.3n_c$
e) $\rightarrow$ f)	0.04 s	(4)	Pledge display	0.04
g) $\rightarrow$ h)	12.5 s	(5a)	Set challenge	0.04
		(5b)	Create ballot	$1.2n_c + 0.6$
			Get ballot from SDCard	$0.04n_c$
	4.4 min			

Table 7.4: Performance times for the registration phase

While some reported to have memorized all three symbols in the “pledge” (29.4%), others memorize just one symbol that they found not to repeat in another position of the ballot (23.5%). Others yet, memorized the candidate which was in front of the “pledge” when the ballot was saved (15.6%). Finally, some memorized the position of the “pledge” in the ballot (13.7%). Note that some voters used several memorization techniques.

Another interesting result was the perception of difficulty of the task; the task is perceived to be much more difficult then it is. While 28.9% of the subjects stated, in the beginning of the experience, that they were expecting to fail (i.e. forgetting the “pledge”), the reality is that only 6.7% (3 subjects) did it. This error in the perception of the difficulty of the task may result from modesty, i.e. the voter may not want to brag about her ability to memorize the code without testing how difficult it is. But it may also result from not perceiving correctly the task being asked. In fact, several voters showed surprised when they were told that they may keep the “pledge” written in the ballot together with the other codes and just have to memorize which of them it is the “pledge”, and may even refresh their memory from time to time, if they want.

Some subjects also reported that they would prefer a different set of symbols, like numbers of letters. In fact, SM&V may be adapted to use several sets of symbols, provided that the voter chooses the set of symbols to use prior to see the pledge (to avoid a covert channel). With such option, one of the set of symbols could be specifically designed for color-blinded voters.

7.2 Performance

From a performance point of view, the voting machine is, also, the critical element, and the registration phase the most critical phase, because it is the one where most cryptographic operations are performed. Recall, that the voting phase requires no more than a signature over a digest of the ballot and a small rotation number. Table 7.4 shows the delay experienced by the voter at the registration phase. The table shows the delays before each screen shown to the user (cf. Figure 4.2), for a 10-candidate election, and the time, per candidate, taken by each operation contributing for those delays. All the measurements were taken with $p = 1536$ bits and $q = 1024$ bits.

The biggest delay is between point & shoot the PDF417 code, with the commitment to the challenge, and the screen asking the voter to tap the PDD (screens c $\rightarrow$ d), which is around 4.2 min. However, this is not the most critical delay given that it may be performed before entering the controlled precinct. The most critical delay is the last one; after point & shoot the QR-code, with the challenge, and being presented with the ballot to vote (screens g $\rightarrow$ h), which is around 12 s. The whole process takes ≈ 4.4 min, although the time that must be spent within the controlled precinct is under 15 seconds, which is reasonable, although it may be further improved with code optimizations and a card featuring modular multiplication [40].

7.3 Security Discussion

The proposed system is resistant to coercion attacks only if the "pledge" is not known by anyone but the voter, and the voter becomes aware of the "pledge" at the same time that it becomes aware of the false "pledges", i.e. the verification codes in the receipt. Which means that the channel between the SE and the voter should not be tappable or interruptible, i.e. the voter should only be allowed to leave the controlled precinct after receiving the receipt, which can be achieved if the last vote registration message may only be sent through the secure environment network. With this assumption the voter may always point to a different code within the receipt, when asked by a coercer, without risk of being caught. Nevertheless, a coercer is still able to force a voter to vote randomly and prevent her from voting.

The system is resistant to a configurable degree of collusion among the participating entities. In fact, the bulletin board, the helper organizations and the electoral commission only handle public information, therefore as long as one HO is honest to help the voter verifying her vote, all the rest may collude without any consequences. However, the number of faulty trustees must be less than t, otherwise the private key of the election is compromised and every vote can be decrypted. This degree of collusion resistance leaves the system less vulnerable against cross-infection between the elements of the model, however some care must be taken with the SE and the PDD. Both elements are in contact with the smartphone, which is assumed to be infected, and if either of these elements gets infected it could compromise the confidentiality of the vote. SM&V relies on the secure design of the secure element and on the simplicity of the PDD to make them immune to smartphone infections.

Nevertheless, even if either the SE, the PDD, or both get infected only the confidentiality of the vote gets affected. If the challenge is correctly generated the secure element can only trick the voter with a probability $1 - p_{soundness}$, $p_{soundness} = (1 - 2^{-\alpha})^{\rho.(n_c-1)}$, which can be set to a configurable low value. Similarly, the probability that the PDD is able to trick the voter in voting for a different candidate is $p_{pdd} = 1 - (1 - 2^{-\alpha})^{(n_c-1)}$, which is the probability that the PDD is able to guess one of the other verification codes.

The system is not resistant to collusion between a coercer and the SE. In fact, if the coercer is able to prevent a voter from complaining about a wrongly casted vote and the SE is compromised, it can submit a vote on any candidate without being detected. Secure elements are designed to be secure, however some are more secure than others. The SE used for the SM&V prototype was a GO-Trust

secure microSD card running Global Platform 2.2.2 and Java 2.2.2 applets.

The solution requires the use of two 2D-codes. The first 2D-code is a commitment for the second 2D-code that encodes the challenge. The goal of the first 2D-code is to prevent a covert channel between the vote and the coercer through the challenge [52]. The 2D-codes may be generated by anyone, including the coercer, provided that it is not colluding with the SE. If the entity generating the 2D-codes and the SE collude it could be possible to change the order of the challenge and commitment to the challenge, which would allow the SE to elude the voter.

The voting application communicates with the PDD using NFC. The NFC technology is particularly suitable for the purpose given that it almost requires contact between the voter's smartphone and the PDD, however, given that, NFC is not currently ubiquitous it is also possible to use Bluetooth or Wifi for this purpose. The use of Bluetooth or Wifi would however preclude the use of standard booth cabinets, given that it will be necessary to prevent the voter's smartphone to communicate with the PDD of a contiguous cabinet. It would also simplify the traceability of the voter's smartphone by the PDD, given that, by opposition to NFC [53], the network address of both Wifi and Bluetooth cannot be random.

The voting machine is the only entity that may not be duplicated or replaced, therefore if the SDCard is broken the voter will be prevented to vote electronically, although, due to the last-resort voting property of SM&V, the voter will always be able to vote physically. All the other entities may either be replaced (e.g. smartphone) or duplicated (e.g. the $\mathcal{BB}$, trustees or helper organizations).

SM&V, as any other network protocol, may be subjected to network infrastructure attacks. Although, the system does not specify any security measures to counteract this type of security attack, it possesses properties that may simplify the design of such security measures. DoS attacks against the election infrastructure with the intent to lock-out one or several electors can be mitigated by replicating the $\mathcal{BB}$. Given that every ballot or vote submission is authenticated and, therefore allowing revoting, most of the problems associated with replication [54] do not exist and replication is easy. Phishing for credentials or vote codes are ineffective in SM&V and therefore do not require special defense mechanisms. In fact, authentication credentials never leave the SE and the votes can only be submitted by someone with the SE. Spoofing the identity (DNS, IP, etc.) of election services is also ineffective provided that at least one helper organization is not spoofed.

Chapter 8

Conclusions

The Internet voting systems are very appealing for the society. Studies shows that these systems are increasing in adoption all over the world [3].

Although, secure mobile Internet elections is a hard goal to achieve, and there is still a long way until all relevant properties are attained simultaneously, I believe SM&V is a step in that direction, mainly because, to my knowledge, it is the first to ensure, simultaneously, a set of security properties (section 4.2) in the remote voting context, without relevant usability degradation.

SM&V assumes that the voter owns a mobile Internet device with a multitouch screen with a secure element (either the UICC, an SDCard or an embed secure element) and proposes a new voting interface with coercion resistant properties that in combination with a variant of the EVIV voting protocol [1] attains also end-to-end verifiability and some resilience to collusion.

In the voting phase the voter is shown two lists side by side on the device. One of the lists contains all the election candidates and the other vote codes. One of these vote codes is correct the others are false. The voter votes by scrolling one or both lists and match her chosen candidate with the correct code.

The proposed system is resistant to coercion attacks only if the “pledge” is not known by anyone but the voter, and the voter becomes aware of the “pledge” at the same time that it becomes aware of the false “pledges”. With this assumption the voter may always point to a different code within the receipt, when asked by a coercer, without risk of being caught. Nevertheless, a coercer is still able to force a voter to vote randomly and prevent her from voting.

In the verification step the voter is able to check that her vote is cast as intended by checking that the “pledge” in her published vote receipt is positioned by the side of her chosen candidate, that the QR-code matches the one used at the time of the voting, and that the ballot receipt is the one presented to the voter.

The SM&V protocol in comparison to EVIV adds the element PDD. The PDD owes its existence to the untrustworthiness of the voter’s smartphone. Being a multipurpose device with many different running applications it is assumed that anything displayed on its screen may be leaked to a coercer. The PDD’s only purpose is to receive, decrypt and display the “pledge” to the voter.

The implementation of SM&V was far from a trivial task. One of the greatest challenges was having to learn to program from beginning for the Android platform. Another obstacle encountered was the process of developing the SM&V applet. The SE used for the prototype has memory limitations, which means that the applet had to be developed optimally to overcome the limitations.

8.1 Future Work

SM&V leaves room for development both in terms of security and usability. From a security point of view the most important evolution would be the resistance to surveillance attacks within the voting booth. This kind of attacks is currently easy to achieve using the camera of a smartphone to record the whole voting process. While eliminating these attacks may be hard, it might be possible to raise the bar for the attacker by incorporating touch sensitive channels (e.g. cold and hot surfaces to report digital bits) between the voter and the SE. How could these be integrated in SM&V and what would be the impact on usability is future work.

From a usability point of view it is also possible to envisioned several developments. The proposed interface does not cope with multiple-choice elections and does not handle well simultaneous elections (requires memorizing several “pledges”). Both kinds of problems may be solved by the same solution given that multiple-choice elections may be mimic as multiple simultaneous elections. Some experiences are currently being made to handle simultaneous elections with just one “pledge”, but they still need some improvements at the user interface level to achieve the desired usability. Another important question to answer is how to cope with blind voters. The interface between the PDD and the voter could be a Braille’s line but the smartphone specific interface needs to be designed and tested.

Bibliography

[1] Joaquim, R., Ribeiro, C., Ferreira, P.: Eviv: an end-to-end verifiable internet voting system. Computers & Security (2012)

[2] Brittain, J.: Clandestine politics and bottom-up organizing in colombia. Journal of Latin American Studies (2005)

[3] Krimmer, R., Triessnig, S., Volkamer, M.: The development of remote e-voting around the world: A review of roads and directions. In: E-Voting and Identity. Volume 4896 of LNCS. October edn. Springer Berlin / Heidelberg, Bochum, Germany (2007) 1–15

[4] Ministry of Local Government and Regional Development: e-vote 2011 - project web site (September 2012) http://www.regjeringen.no/en/dep/krd/prosjekter/e-vote-2011-project.html?id=597658.

[5] Rubin, A.D.: Security considerations for remote electronic voting. Commun. ACM **45**(12) (December 2002) 39–44

[6] Oostveen, A.M., Van den Besselaar, P.: Security as belief: user?s perceptions on the security of electronic voting systems. Electronic voting in Europe: Technology, law, politics and society **47** (2004) 73–82

[7] Neff, C.A.: Practical high certainty intent verification for encrypted votes. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.134.1006 (2004) http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.134.1006.

[8] Chaum, D.: Secret-ballot receipts: True voter-verifiable elections. IEEE Security and Privacy **2** (2004) 38–47

[9] Clarkson, M., Chong, S., Myers, A.: Civitas: Toward a secure voting system. In: IEEE Symposium on Security and Privacy, Oakland, CA, USA, IEEE Computer Society (May 2008) 354–368

[10] Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Proc. of the 2005 ACM workshop on Privacy in the electronic society, Alexandria, VA, USA, ACM (November 2005) 61–70

[11] Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques. EUROCRYPT'00, Berlin, Heidelberg, Springer-Verlag (2000) 539–556

[12] ElGamal, T.: A public-key cryptosystem and signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **IT-31**(4) (July 1985) 469–472

[13] Schoenmakers, R.: A secure and optimally efficient multi-authority election scheme. Transactions on Emerging Telecommunications Technologies 8 (1997)

[14] Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2) (1981) 84–90

[15] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: STOC. (1985) 291–304

[16] Chaum, D.: Blind signatures for untraceable payments. In: CRYPTO. (1982) 199–203

[17] Shamir, A.: How to share a secret. Commun. ACM **22**(11) (1979) 612–613

[18] Pedersen, T.P.: A threshold cryptosystem without a trusted party (extended abstract). In: EUROCRYPT. (1991) 522–526

[19] Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: EUROCRYPT '97. Volume 1233 of LNCS. Springer Berlin / Heidelberg, Konstanz, Germany (1997) 103–118

[20] Joaquim, R., Ribeiro, C.: An efficient and highly sound voter verification technique and its implementation. E-Voting and Identity (2012) 104–121

[21] Adida, B., Neff, A.: Efficient receipt-free ballot casting resistant to covert channels. In: USENIX EVT/WOTE. (2009)

[22] Chaum, D., Pedersen, T.: Wallet databases with observers. In: CRYPTO '92. Volume 740 of LNCS., Springer (1992) 89–105

[23] Cranor, L.F., Cytron, R.K.: Sensus: A security-conscious electronic polling system for the internet. (1997) 561–570

[24] Kiayias, A., Korman, M., Walluck, D.: An internet voting system supporting user privacy. In: ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference, IEEE Computer Society (2006) 165–174

[25] Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: AUSCRYPT. (1992) 244–251

[26] Joaquim, R., Zúquete, A., Ferreira, P.: Revs: A robust electronic voting system. IADIS - International Journal of WWW/Internet (December 2003)

[27] Estonian National Electoral Commitee: Internet voting in estonia (April 2012) http://www.vvk.ee/voting-methods-in-estonia/engindex/.

[28] Gjosteen, K.: The norwegian internet voting protocol. In Kiayias, A., Lipmaa, H., eds.: E-Voting and Identity. Volume 7187 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2012) 1–18

[29] Helios: Helios web site (2011) http://heliosvoting.org/.

[30] Kutyłowski, M., Zagórski, F.: Scratch, click and vote: E2e voting over the internet. In: Towards Trustworthy Elections. Volume 6000 of LNCS. Springer (2010) 343–356

[31] Joaquim, R., Ribeiro, C., Ferreira, P.: Veryvote: A voter verifiable code voting system. In: E-Voting and Identity. Volume 5767 of LNCS. Springer (2009) 106–121

[32] Adida, B.: Helios: web-based open-audit voting. In: SS'08: Proceedings of the 17th USENIX Security symposium, Berkeley, CA, USA, USENIX Association (2008) 335–348

[33] Benaloh, J.: Simple verifiable elections. In: EVT 2006, Berkeley, CA, USA, USENIX Association (2006)

[34] Heather, J., Lundin, D.: The append-only web bulletin board. In: Formal Aspects in Security and Trust. Springer, Eindhoven, The Netherlands (November 2009) 242–256

[35] Nohl, K.: Rooting sim cards. In: BlackHat US 2013, Las Vegas, NV (July 2013)

[36] International Organisation for Standardization (ISO): Telecommunications and information exchange between systems - near field communication - interface and protocol. Technical Report ISO/IEC 18092 - Information technology, ISO (April 2004)

[37] Global Platform: Card specification v2. 1.1. Technical report, Online: http://www. globalplatform. org (2003)

[38] Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Advances in Cryptology—EUROCRYPT'91, Brighton, UK, Springer (April 1991) 522–526

[39] Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Advances in Cryptology—CRYPTO'89 Proceedings, Santa Barbara, CA, USA, Springer (1990) 307–315

[40] Joaquim, R., Ribeiro, C.: An efficient and highly sound voter verification technique and its implementation. In Kiayias, A., Lipmaa, H., eds.: E-Voting and Identity. Volume 7187 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Tallinn, Estonia (2012) 104–121

[41] Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: CRYPTO '94. Volume 839 of LNCS., Santa Barbara, CA, USA, Springer (1994) 174–187

[42] : http://dev.mysql.com/downloads/mysql/5.5.html.

[43] : https://code.google.com/p/ksoap2-android/.

[44] Alliance, S.: Open mobile api specification. Technical Report 2.0.2, SIM Alliance (November 2011)

[45] : http://www.gemalto.com/products/Developer_Suite/.

[46] : http://www.go-trust.com/products/microsd-java/.

[47] : https://github.com/PDF417/pdf417-android/.

[48] : http://rtyley.github.io/spongycastle/.

[49] : https://github.com/zxing/zxing.

[50] Bertin, J.: Semiology of graphics: diagrams, networks, maps. University of Wisconsin press (1983)

[51] Craik, F.I., Lockhart, R.S.: Levels of processing: A framework for memory research. Journal of verbal learning and verbal behavior **11**(6) (1972) 671–684

[52] Adida, B., Neff, C.A.: Ballot casting assurance. In: EVT 2006, Berkeley, CA, USA, USENIX Association (2006) 7–15

[53] ETSI: Smart cards; uicc - contactless front-end (clf) interface; host controller interface (hci). Technical Report ETSI TS 102 622, European Telecommunications Standards Institute (2008)

[54] Dini, G.: A secure and available electronic voting service for a large-scale distributed system. Future Generation Computer Systems **19**(1) (2003) 69 – 85

[55] Chor, B., Goldreich, O., Hasted, J., Freidmann, J., Rudich, S., Smolensky, R.: The bit extraction problem or t-resilient functions. In: 26th Annual Symposium on Foundations of Computer Science, NJ, USA, IEEE Computer Society (1985) 396–407

[56] Kurosawa, K., Johansson, T., Stinson, D.R.: Almost k-wise independent sample spaces and their cryptologic applications. Journal of Cryptology **14**(4) (2001) 231–253

[57] Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Towards Trustworthy Elections. Springer, New York, NY, USA (2010) 37–63

Appendix A

Appendix: SM&V Integrity and Confidentiality Proofs

A.1 Integrity Properties

The integrity properties P_{I1}, P_{I2} and P_{I3} result directly from EVIV's [1] properties $P1_{EVIV}$, $P2_{EVIV}$ and $P3_{EVIV}$, provided that the challenge $chal_i^n$ is fresh, random and independently generated from the voting machine. In EVIV there is a single challenge for all the voters generated by a group of trustees, but in SM&V that option is not available given that every ballot must be open in different moments, therefore with different challenges.

Lemma A.1.1. *The challenge $chal_i^n$ is randomly and independently chosen from the voting machine running in the SE.*

Proof. Under the assumption that the SE does not collude with whomever generates the QR-Code r_i, then r_i is independently chosen from the voting machine. If $H(\cdot)$ is a k-resilient hash function [55] (or an almost k-resilient hash function [56]) the generated challenge $chal_i^n = H(r_i, BallotReceipt_i, n) \bmod q$ is also independently chosen from the voting machine. □

Lemma A.1.2. *The challenge $chal_i^n$ is freshly generated for each ballot under the random oracle model.*

Proof. It follows directly from $chal_i^n = H(r_i, BallotReceipt_i, n) \bmod q$ and $BallotReceipt_i = H(\langle k, ClosedBallot_i \rangle_{sk_{\mathcal{BB}}})$ be committed to the voter before the pledge being displayed. □

A.2 Privacy Proofs

Coercion resistance is a generalization of the basic privacy property. Privacy in a voting system is usually defined in terms of an adversary that cannot interact with voters during the election process. Therefore, by proving that under the specified assumptions SM&V satisfies property $(\mathbf{P_{C2}})$ (Coercion resistance) it is also being proved that it satisfies property $(\mathbf{P_{C1}})$.

Theorem A.2.1. $(\mathbf{P_{C2}})$ *Voters cannot prove how they voted, even if they can interact with adversary while voting.*

Proof. Our prove follows the Juels *at al.* approach [57], which defines the property of coercion-resistance by an experiment $\mathbf{Exp}_{\mathcal{A}}^{\textit{c-resist}}$ involving an adversary $\mathcal{A}$ in interaction with the SM&V system. The experiment output is always '1' or '0'; the aim of the adversary is to cause the experiment to output '1' with a probability non negligible, in the standard cryptographic meaning of negligible.

The experiment features a game between $\mathcal{A}$ and a voter targeted by the adversary (denoted by voter $\mathcal{V}_\sigma$) for coercive attack. Within the game, the coerced voter flips a coin and sets the bit b with the result. If $b = 1$ then the voter $\mathcal{V}_\sigma$ submits to the coercer and provides a valid voting credential; if $b = 0$ then the voter provides the coercer with a false, but indistinct voting credential. The false credential provided by $\mathcal{V}_\sigma$ varies with the instant of coercion. We model the behavior of $\mathcal{V}_\sigma$ by a function ψ of her knowledge of π_σ and $\vartheta_\sigma = \{\vartheta_{\sigma,j}\}_{j=0}^{n_c}$ at each point of the simulation

$$\psi_\sigma(b, \pi_\sigma, \vartheta_\sigma) = \begin{cases} \perp & \text{if } \pi_\sigma = \perp \\ \pi_\sigma & \text{if } \pi_\sigma \neq \perp \text{ and } b = 1 \\ x \in_R \vartheta_\sigma : x \neq \pi_\sigma & \text{if } \pi_\sigma \neq \perp \text{ and } b = 0 \text{ and } \vartheta_\sigma \neq \perp \\ x \in \mathbb{Z}_q & \text{if } \pi_\sigma \neq \perp \text{ and } b = 0 \text{ and } \vartheta_\sigma = \perp \end{cases}$$

where $\pi_\sigma = \perp$ and $\vartheta_\sigma = \perp$ means that the voter has no knowledge of π_σ and ϑ_σ at that time. Notice that, if $\mathcal{V}_\sigma$ tries to deceive the adversary ($b = 0$), when already knows the pledge π_σ but has no knowledge of the false credentials ϑ_σ, she will reply with a random number ($x \in_R \mathbb{Z}_q$). However, in such scenario she will be caught by the adversary if, whenever ϑ_σ are eventually revealed, none of them matches x. Therefore, such situation should never occur in our game.

The task of the adversary is to guess the value of b, i.e. to determine if the voter gave the correct voting credential or a false one. Given that, at the time of the voting, the adversary has access to every credential ϑ_σ, he may choose the one provided by the voter $\psi_\sigma(b, \pi_\sigma, \vartheta_\sigma)$ or a different one, according with some previous acquired knowledge D_{π_σ} about voters' intentions. This information may be acquired by surfer shouldering the voter or by knowing in which candidate would the voter vote if not coerced. We further characterize the adversary by assuming that he controls a limited number of voters $V, |V| \leq n_v$.

The experiment makes use of four voting functions: `regClose()`, `regOpen()`, `vote()` and `tally()`, representing respectively the registration of the closed and open ballots of every voter, the vote casting of all honest voters (i.e. not corrupted and not being coerced) and the tabulation of all casted votes.

The closed ballot registration function $\texttt{regClose}(\mathcal{V}_i, sk_i, pk_{\mathcal{T}}, n_c)$ takes the voter private key sk_i, the election public key $pk_{\mathcal{T}}$, the number of candidates n_c, and outputs a close ballot $CloseBallot_i$ and a voting credential π_i ("pledge") for each voter.

The registration function $\texttt{regOpen}(\mathcal{V}_i, sk_i, sk_{\mathcal{BB}}, BallotReceipt_i, n_c, r_i)$ takes the voter and the $\mathcal{BB}$ private keys $(sk_i, sk_{\mathcal{BB}})$, the $BallotReceipt_i$ received from the $\mathcal{BB}$, and the challenge r_i encoded in the voter's QR-code, and outputs $Ballot_i$ and all the remaining voter credentials ϑ_i for each voter $\mathcal{V}_i$.

The vote casting function $\texttt{vote}\,(\pi_i, sk_i, D_{\mathcal{C},\mathcal{V}})$ takes the credentials π_i from every honest voter, their

```
Experiment Exp^{c-resist}_{A}(C, V, n_v, n_c, n_σ)
V ← A(C, "Control votes")                                              // A corrupts voters
(σ, c_σ) ← A(n_v, n_c, "Set target voter and vote")                    // A sets coercive target
b ∈_R {0, 1};                                                          // Coin is flipped
{(r_i, c_{r_i}) ← A(ψ_σ(b, ⊥, ⊥), "Generates 2D-codes")}_{i=1}^{n_v}   // A generates the QR-Codes
{(ClosedBallot_i, π_i) ← regClose(V_i, sk_i, pk_T, n_c)}_{i=1}^{n_v}   // SE generates Closed Ballots and "pledges"
{BB ⇐ c_{r_i}, ClosedBallot_i}_{i=1}^{n_v}                             // Closed Ballots are published
{BallotReceipt_i ← BB}_{i=1}^{n_v}                                     // Ballot receipts are generated
if |V| ≠ n_σ or σ ∈ V or σ > n_v or
   c_σ > n_c or c_{r_i} ≠ H(r_i) then                                  // Outputs of A checked for validity
      output '0';
{(Ballot_i, ϑ_i) ← regOpen(i, sk_i, sk_BB, BallotReceipt_i, r_i)}_{i=1}^{n_v}   // SE generates ballots
{BB ⇐ Ballot_i}_{i=1}^{n_v}                                            // SMV Ballots are published
{BB ⇐ vote(π_i, sk_i, D_{C,V}}_{i≠σ, i∉V}                              // Ballots post by honest voters
π̂_σ ← A(ψ_σ(b, π_σ, ϑ_σ), ϑ_σ, D_{π_σ}, "Get Credential")             // A chooses a credential
BB ⇐ A({π̂_σ or ∅}, sk_i, BB, "Cast ballots")                           // A posts to BB
Γ ← tally(sk_T, BB, n_c)                                               // Elections result are tallied
b' ← A(Γ, BB, D_{π_σ}, "Guess b")                                      // A guesses coin flip
if b = b' then                                                         // Experimental output determined
   output '1';
else
   output '0';
```

Figure A.1: Coercion resistant experiment under an adversary $\mathcal{A}$.

private keys sk_i, and a probability distribution $D_{\mathcal{C},\mathcal{V}}$ denoting honest voters voting pattern, and outputs the votes.

Finally, the tally function $\texttt{tally}\,(sk_{\mathcal{T}}, \mathcal{BB}, n_c)$ takes all the information stored in the $\mathcal{BB}$ and the election private key $sk_{\mathcal{T}}$ and outputs the tabulation result Γ of the election.

The experiment $\textbf{Exp}^{\textit{c-resist}}_{\mathcal{A}}$ in Figure A.1 describes formally the coercion resistant property satisfied by SM&V, using the above described functions and a couple of operators: $\leftarrow$ denotes the assignment operator and $\Leftarrow$ the append operator.

The adversary $\mathcal{A}$ in experiment $\textbf{Exp}^{\textit{c-resist}}_{\mathcal{A}}$ has access to all the experiment data apart from the b and π_i of each non corrupted voter. Our task is to show that all these data provide no extra advantage to $\mathcal{A}$ in guessing bit b, when compared with an adversary $\mathcal{A}'$ that has access only to the tabulation result. This adversary $\mathcal{A}'$ and experiment $\textbf{Exp}^{\textit{c-resist-ideal}}_{\mathcal{A}'}$ are described in Figure A.2.[1]

The main difference between the two experiments is that, in the latter, the adversary only gets to know Γ - the tabulation result, and L - the set of absentees. Ideally, L would not be available to the coercer, but SM&V is not able to hide that information. In fact, SM&V uses such limitation to allow for a last resort presential voting, as it is done in Estonia and the Norway [4] electronic voting systems, i.e. the voter may always revoke her e-vote and choose to vote presentially.

As in Juels *at al.* [57], our proof strategy is to construct a polynomial-time algorithm $\mathcal{S}$ that simulates the election system SM&V in the c-resist experiment, which is perfectly indistinguishable to $\mathcal{A}$ from using true building blocks of SM&V and, that cannot be deviated from the correct execution by $\mathcal{A}$.

The inability of the adversary to influence the correct execution of the Simulation results from the ability to public verify the random commitments and the Zero-Knowledge proofs modeled by the oracles $\Omega(cv)$ and $\Omega(v)$ and the MP3 ZK receipt creation. Being publicly verifiable everyone may check the

[1] Grey lines are unchanged from experiment $\textbf{Exp}^{\textit{c-resist}}_{\mathcal{A}}$.

1 Experiment $\mathbf{Exp}^{\textit{c-resist-ideal}}_{\mathcal{A}'}(\mathcal{C}, \mathcal{V}, n_v, n_c, n_a)$

2 $V \leftarrow \mathcal{A}'(\mathcal{C}, \texttt{"Control votes"})$ // $\mathcal{A}'$ corrupts voters

3 $(\sigma, c_\sigma) \leftarrow \mathcal{A}'(n_v, n_c, \texttt{"Set target voter and vote"})$ // $\mathcal{A}'$ sets coercive target

4 $b \in_R \{0,1\}$; // Coin is flipped

5 $\{(r_i, c_{r_i}) \leftarrow \mathcal{A}'(\psi_n u(b, \perp, \perp), \texttt{Generates 2D-codes})\}_{i=1}^{n_v}$ // Generates the QR-Code

6 $\{(ClosedBallot_i, \pi_i) \leftarrow \texttt{regClose}(i, sk_i, pk_\mathcal{T}, n_c)\}_{i=1}^{n_v}$ // SE generates Closed Ballots and pledges

7 $\{\mathcal{BB} \Leftarrow c_{r_i}, \textit{ClosedBallot}_i\}_{i=1}^{n_v}$ // Closed Ballots are published

8 $\{\textit{BallotReceipt}_i \leftarrow \mathcal{BB}\}_{i=1}^{n_v}$ // Ballot receipts are generated

9 **if** $|V| \neq n_v$ or $\sigma \in V$ or $\sigma > n_v$ or

10 $c_\sigma > n_c$ or $c_{r_i} \neq H(r_i)$ **then** // Outputs of $\mathcal{A}$ checked for validity

11 **output** '0';

12 $\{(\textit{Ballot}_i, \vartheta_i) \leftarrow \texttt{regOpen}(i, sk_i, sk_{\mathcal{BB}}, \textit{BallotReceipt}_i, r_i)\}_{i=1}^{n_v}$ // SE generates SMV ballot

13 $\{\mathcal{BB} \Leftarrow \textit{Ballot}_i\}_{i=1}^{n_v}$ // SMV Ballots are published

14 $\{\mathcal{BB} \Leftarrow \texttt{vote}(\pi_i, sk_i, n_c, D_{\mathcal{C},\mathcal{V}})\}_{i \neq \sigma, i \notin V}$ // Ballots posted by honest voters

15 $\hat{\pi}_\sigma \leftarrow \mathcal{A}'(\psi_\sigma(b, \pi_\sigma, \vartheta_\sigma), \vartheta_n u, D_{\pi_\sigma}, \texttt{"Get Credential"})$ // $\mathcal{A}'$ chooses a credential

16 $\mathcal{BB} \Leftarrow \mathcal{A}'(\{\hat{\pi}_\sigma \text{ or } \emptyset\}, sk_i, \mathcal{BB}, \texttt{"Cast ballots"})$ // $\mathcal{A}'$ posts to $\mathcal{BB}$

17 $(\Gamma, L) \leftarrow \texttt{tally}(sk_\mathcal{T}, \mathcal{BB}, n_c)$ // Elections result are tallied

18 $b' \leftarrow \mathcal{A}'(\Gamma, L, D_{\pi_\sigma}, \texttt{"guess b"})$ // $\mathcal{A}'$ guesses coin flip

19 **if** $b = b'$ **then** // Experiment output determined

20 **output** '1';

21 **else**

22 **output** '0';

Figure A.2: Coercion resistant experiment under an adversary $\mathcal{A}'$.

actions of every actor in the process.

We then show that, under the Decisional Diffie Hellman assumption, the simulator reveals nothing to the adversary but the election tally Γ and the list of voter absentees L.

We delineate the simulator $\mathcal{S}$ along the lines of experiment $\mathbf{Exp}^{\textit{c-resist}}_{\mathcal{A}}$ using the building blocks of SM&V to implement `regClose()`, `regOpen()`, `vote()` and `tally()` functions. Each step of the simulator described below is marked with the correspondent line numbers in Figure A.1.

The simulator input is a tuple (g_1, g_2, g_3, g_4) which is either a Diffie-Hellman quadruple or a random one, according to some hidden bit d. We set (g_1, g_2, g_3) to be always $g_1 = g, g_2 = g^x, g_3 = g^y$ where $x, y \in_R \mathbb{Z}_q$ and g is a generator of the $\mathbb{Z}_p*$ subgroup G_q of order q, where p and q are large primes such that $q|p-1$. And then set g_4 to be either $g_4 = g^{xy}$ if $d = 1$ or $g_4 = g^z$ for some $z \in_R \mathbb{Z}_q$, if $d = 0$. The goal of the simulator is to distinguish between the two input situations.

Setup $\mathcal{S}$ starts by choosing the election public key $pk_\mathcal{T} = h = g^y$ and publishing it along with the candidate list $\mathcal{C}$ in the $\mathcal{BB}$.

Adversarial corruption (lines:2-3) The adversary $\mathcal{A}$ selects a set V of n_σ voters to corrupt and chooses the voter σ to coerce. If some of the choices are invalid the simulation stops.

Coin flip (line:4) A coin $b \in_R \{0,1\}$ is flipped.

Setup Coercion (line:5) The adversary $\mathcal{A}$ generates $r_i = r \parallel \psi_\sigma(b, \pi_\sigma, \vartheta_\sigma)$, where $r \in_R \mathbb{Z}_q$, and a commitment $c_{r_i} = H(r_i)$. Given that at this time $\pi_\sigma = \vartheta_\sigma = \perp$ then $\psi_\sigma(b, \pi_\sigma, \vartheta_\sigma) = \perp$ and $r_i = r$.

Close Ballot registration (lines:6-7) In the closed ballot registration step, $\mathcal{S}$ generates the "pledge" $\pi_i \in_R \mathbb{Z}_q$ and several random numbers $\tau_{ij}, \delta_{ij}, \theta_{ij} \in_R \mathbb{Z}_q$ and then generates a $CloseBallot_i$

using the simulation input (g_1, g_2, g_3, g_4).

$$CloseBallot_i = \{(g_2^{\tau_{ij}}, g_1^{m_{ij}} g_4^{\tau_{ij}}), (g_2^{\delta_{ij}}, g_1^{\theta_{ij}} g_4^{\delta_{ij}})\}_{j=1}^{n_c}$$

$$m_{ij} = \begin{cases} -1 & j \neq p_i \\ 1 & j = p_i \end{cases}$$

$$\pi_i = \theta_{ij}|_{j=p_i}$$

Receipt Generation (line:8) The receipt generation step takes the $CloseBallot_i$ and the commitment to the challenge c_{r_i} and generates a receipt for the Close Ballot by signing it with the $\mathcal{BB}$ key.

$$BallotReceipt_i = \langle c_{r_i}, CloseBallot_i \rangle_{sk_{\mathcal{BB}}}$$

Registration (lines:12-13)**:** In the registration step, $\mathcal{S}$ takes the $CloseBallot_i$ and the random number produced by the adversary r_i to generate the ballot $Ballot_i$

$$\textit{Ballot}_i = \langle \mathcal{V}_i, BallotReceipt_i, CloseBallot_i, chal_i^n, \{\vartheta_{ij}, \omega_{ij}\}_{j=1}^{n_c}, r_i, c_{r_i} \rangle_{sk_i}$$

$$chal_i^n = H(r_i, BallotReceipt_i, n) \bmod q, \quad 0 < n \leq \rho$$

$$\vartheta_{ij} = \begin{cases} 2.chal_i^n - \theta_{ij} \bmod q & j \neq p_i \\ \theta_{ij} & j = p_i \end{cases}$$

$$\omega_{ij} = \tau_{ij} \cdot (chal_i^n - \vartheta_{ij}) + \delta_{ij} \bmod q$$

Honest voter simulation (line:14) For each ballot *Ballot*$_i$ such that $i \neq \sigma$ and $i \notin V$ the simulator generates a vote *Vote*$_i$

$$\textit{Vote}_i = \langle H(\textit{Ballot}_i), rot_i \rangle_{sk_i}$$

$$rot_i = \begin{cases} c_i - p_i \bmod n_c & \text{if } i \text{ casts a vote} \\ \emptyset & \text{if } i \text{ abstains} \end{cases}$$

where c_i is the index of the chosen candidate and p_i is the "pledge" position in the receipt.

Credential release (line:15) $\mathcal{S}$ gives $\mathcal{A}$ the set of credentials ϑ_σ and a credential resulting from the evaluation of $\psi_\sigma(b, \pi_\sigma, \vartheta_\sigma)$, which $\mathcal{A}$ uses to guess $\hat{\pi}_\sigma$ using the knowledge on the voters history D_{π_σ}.

Adversarial ballot posting (l:16) The adversarial $\mathcal{A}$ posts a set of ballots *Ballot*$_i$ such that $i = \sigma$ or $i \in V$.

Tallying simulation (line:17) The tabulation of the votes proceeds as follows:

- The Simulator verifies the signatures on all the votes and ballots and checks that each casted vote matches the previous registered ballot. It also checks the validity of each random value, i.e. $c_{r_i} = H(r_i)$, $n < \rho$ and that $chal_i^n$ was correctly generated. Every non complying vote is removed from the tally.

- Then the Simulator uses oracles $\Omega(cv)$ and $\Omega(v)$ to verify if every ballot is well-formed, i.e. if it is comprised of one Exponential ElGamal encryption of '1' and $n_c - 1$ Exponential ElGamal encryptions of '-1'. Every vote and ballot not complying is removed from the tally.
- The last verification step is to verify if the MP receipt is correct by checking the MP ZK proof. Again every non complying vote is removed from the tally
- Finally, all valid votes get homomorphically counted and decrypted by the trustees. The trustees also produce a ZK proof of correct tally decryption.

Given that rot_σ, $chal_\sigma^n$, $BallotReceipt_\sigma$ or ϑ_σ trivially reveals anything about b or π_σ, and that r_σ and c_{r_σ} are generated without any knowledge of b or π_σ (recall that $\psi(b, \perp, \perp) = \perp$, line 5) then if the simulator's input is a Diffie-Hellman tuple ($d = 1$), the simulation is indistinguishable from $\textbf{Exp}_{\mathcal{A}}^{\textit{c-resist}}$. Assuming that the DH tuple is given by $(g_1 = g, g_2 = g^x, g_3 = g^y, g_4 = g^{xy})$, then cv_i denotes a set of DH tuples $\{(g^{x\tau_{ij}}, g^{m_{ij}} h^{x\tau_{ij}}), (g^{x\delta_{ij}}, g^{\theta_{ij}} h^{x\delta_{ij}})\}_{j=1}^{n_c}$, and the probability of the simulator being able to output '1' (i.e. guessing the bit b) is equal to the probability of $\textbf{Exp}_{\mathcal{A}}^{c-resist}$ to be succeeded (i.e. to output '1')

$$Pr[\mathcal{S} = 1 | d = 1] = Pr[\textbf{Exp}_{\mathcal{A}}^{c-resist}() = 1]$$

If the simulator's input is not a Diffiel-Helman tuple ($d = 0$) then the simulation is indistinguishable from $\textbf{Exp}_{\mathcal{A}'}^{c-resist-ideal}$. In fact, assuming that the simulator's input is given by $(g_1 = g, g_2 = g^x, g_3 = g^y, g_4 = g^z)$, then $cv_i = \{(g^{x\tau_{ij}}, g^{m_{ij}} h^{x\tau_{ij}} g^{z'\tau_{ij}}), (g^{x\delta_{ij}}, g^{\theta_{ij}} h^{x\delta_{ij}} g^{z'\delta_{ij}})\}_{j=1}^{n_c}$, which would perfectly hide $g^{m_{ij}}$ and $g^{\theta_{ij}}$ if $g^{z'\tau_{ij}}$ and $g^{z'\delta_{ij}}$ where completely random and independent numbers. Given that τ_{ij} and δ_{ij} are random numbers related by $\omega_{ij} = \tau_{ij} \cdot (chal_i - \vartheta_{ij}) + \delta_{ij} \bmod q$ then τ_{ij} and δ_{ij} are not independent. However, given that the simulator tally removed any votes not satisfying equation 5.2 the following equality holds

$$(g^{x\omega_{ij}}, g^{chal_i} g^{z\omega_{ij}}) = (g^{x\tau_{ij}}, g^{m_{ij}} g^{z\tau_{ij}})^{chal_i - \vartheta_{ij}} \cdot (g^{x\delta_{ij}}, g^{\theta_{ij}} g^{z\delta_{ij}})$$

that for any given $chal_i, \tau_{ij}, \delta_{ij}$ and θ_{ij} has only 2 equally probable solutions:

$$m_{ij} = \begin{cases} 1 & \vartheta_{ij} = \theta_{ij} \\ -1 & \vartheta_{ij} = 2.chal_i - \theta_{ij} \bmod q \end{cases}$$

which reveals nothing about b, therefore the probability of the Simulator to guess b when $d = 0$ is equal to the probability that the adversary outputs 1 in experiment $\textbf{Exp}_{\mathcal{A}'}^{c-resist-ideal}$.

$$Pr[\mathcal{S} = 1 | d = 0] = Pr[\textbf{Exp}_{\mathcal{A}'}^{c-resist-ideal}() = 1]$$

which means that the advantage of the simulator in breaking DDH is equal to the advantage of the experiment $\textbf{Exp}_{\mathcal{A}}^{c-resist}$,

$$\textbf{Adv}_{\mathcal{S}}^{DDH} = Pr[\mathcal{S} = 1 | d = 0] - Pr[\mathcal{S} = 1 | d = 1] = \textbf{Adv}_{\mathcal{A}}^{c-resist}$$

which is negligible under the Decisional Diffie Hellman Assumption □ □

Notice that, under the simulator, and both experiments, the coerced voter never re-votes. In fact, if the coerced voter re-votes that is immediately detected by the coercer. In such a situation the coercer, besides the information in the $\mathcal{BB}$, would have an additional rot_σ value different from the one in the $\mathcal{BB}$. This extra information would be enough to guess b.