Paper-based electronic voting

Anna Solveig Julia
Testaniere

Master of Science in Mathematics
Submission date: December 2015
Supervisor: Kristian Gjøsteen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Paper-based electronic voting

Anna Solveig Julia Testanière

December 2015

Table of Contents

Abstract

In this thesis, we present two paper-based electronic voting systems Prêt-à-Voter and Demos. We describe these in the same systematic way with new examples. Furthermore, we implement RSA cryptosystem in Prêt-à-Voter. Then, we propose an informal analysis of what is required both in practice and in the technical part of Prêt-à-Voter. We present randomized partial checking during mix-net and emphasize issues surrounding this component based on Pfitzmann attack and duplicate a vote attack. We discuss the size of a ciphertext in Prêt-à-Voter and explain the difficulty of proving permutation and randomness in the system. Finally, we discuss the concept of privacy based on the privacy game and illustrate with attacks how the privacy can be broken in both Prêt-à-Voter and Demos. For a general analysis of these two voting systems this thesis should be read together with the thesis of Anna Vederhus.

Acknowledgement

First of all, I would like to thank Kristian Gjøsteen for his incredible knowledge in his field and his interest in voting systems. I also thank Martin Strand for his guidance during the spring semester. I thank my mother for her tremendous support and her help during these last five years. I am very grateful for her, my father, Karina and Doriane and for their love and faith in me. I thank the students from room 395B and especially Sigurd, Marit and Anna for their friendship, talent in ping-pong, sense of humor and participation in every kind of discussion. I thank Dag-Vidar Bauer for his love and patience during these five years. I could not have done it without him. Finally, I would like to thank Anna Vederhus for such a good cooperation in mathematics during this very intense year of work and for our wonderful friendship that emerged from it. You are really one of a kind.

Chapter 1

Introduction

In an election, it is crucial for the society to trust the voting system implemented. That is, the voter should be certain that when the election is done her vote is counted as intended. This property is called verifiability. Moreover, she should be able to vote as she wants without being controlled by another, and she should have the right to vote privately. This is the privacy property. Finally, the system should be accessible and understandable to every voter. This is the availability property.

These requirements, categorised into verifiability, privacy and availability are fundamental in a trustworthy system. Unfortunately, it is a quite difficult task to satisfy these properties simultaneously. Indeed, it seems that more of one gives less of the other. For example, if the voting system allows the voter to use her personal mobile-phone as a voting device, the system is definitely available, but the privacy of the vote can not be assured.

Let's now have a closer look at our traditional way of voting. In Norway, the voter must go to a polling place. There, she makes her choice in a polling booth, privately. Her paper ballot is folded so that no one can see how she voted. Then, she must register and prove to the authority that she is eligible to vote. If yes, her ballot is stamped and she puts her ballot in an urn, which will be counted at the end of the election by the authority. It is a well-thought-out system, but can we really trust this traditional way of voting? Does it fulfill the trust-requirements that we have stated?

Firstly, this system appears available since the way of voting is understandable to all and the polling places are accessible to some extent. Secondly, it seems to ensure privacy as the voter is alone in the voting booth and leaves without any receipt. But, let's not forget that nowadays the voter can use her personal electronic devices while voting and take pictures of or film her choices. This can cause vote-selling and making it possible for an adversary to coerce her.

Finally, the voter can not be certain that her ballot has been counted or even counted correctly. In other words, the voter must trust the election authority and the tellers to be honest while tallying her and all the other ballots. Miscounting and cheating occurs more often than we think and can have great consequences. So, as we can see, this voting system has some weaknesses motivating research of a better solution. Can we make a system more secure, robust and trustworthy using new technology?

In an electronic voting system, cryptography could offer more security for instance by making it possible for the voter to verify her vote and making it more difficult for the different agents involved in a voting system to cheat. Moreover, by avoiding the human counting, the tallying procedure could be more efficient. This could also reduce the cost of the election, especially if we consider remote electronic voting. Although not important for a trustworthy system, efficiency and cost are properties that must be taken into consideration.

However, turning to electronic voting does not only come with advantages. If we consider a remote electronic voting system, where the voter can cast her ballot from home, at work or in the bus, how can we ensure the privacy needed? How can we be certain that she was not forced or manipulated to vote in some way? This problem is so large and complicated that as per today, a remote system that ensures privacy, has not been found. We need the privacy of the voter to be kept, hence it has to be supervised, but we also want the voter to be able to verify her vote, therefore we turn to supervised electronic voting. By that, we mean having cryptographic elements making the verification possible, without losing the privacy requirement as the voting is still happening at a polling station.

Table Three voting systems with a superficial analysis of benefits and drawbacks.

	Availability	**Privacy**	**Verifiability**
Traditional Norwegian voting system	Yes. - **Accessible** to the extent that the voter needs to go to a polling station to vote. - **Understandable** to every voter.	Yes. - **Coercion resistant** because of polling booth and receipt-freeness. - **Anonymous**.	No. - The voter cannot verify that her vote was counted. - Only parts of the election process can be verified.
Remote voting system	Yes. - **Accessible** to the extent that the voter has access to Internet. - **Understandable** to every voter, may be more complicated because of the cryptographic elements and the use of the technological devices.	Difficult. - **No privacy** ensured while voting, more complicated to achieve coercion resistance. - **Anonymous** to the extent that her vote can not be traced in the system.	Yes. - The voter can verify that her vote was counted. - All parts of the election process can be verified.
Paper-based electronic voting system	Yes. - **Accessible** to the extent that the voter needs to go to a polling station to vote. - **Understandable** to every voter, may be more complicated because of the cryptographic elements.	Yes. - **Coercion resistant** because of polling booth and cryptographic receipt. - **Anonymous**.	Yes. - The voter can verify that her vote was counted. - All parts of the election process can be verified.

In chapter 2, we present the definitions and mathematical notions used in this thesis. In chapter 3, we describe the voting system Demos and explain further the setup phase, the voting phase and the tallying phase of the system. We conclude this chapter with an illustrating example. Similarly, we present in chapter 4 two versions of the voting system Prêt-à-Voter. First, we describe a version based on decryption mix-net and second, a version based on re-encryption mix-net. We also make an illustrating example of the first version.

We analyse Prêt-à-Voter with respect to privacy, verifiability and availability in chapter 5 and 6. We start by analysing the voter verifiability in chapter 5 and continue our analysis by investigating the cryptographic components in chapter 6. Finally, we discuss the concept of privacy and introduce possible attacks in Prêt-à-Voter and Demos in chapter 7.

Chapter 2

Theory

2.1 Definitions

Numerous notions and many requirements can be used to define a voting system. We define only the notions and requirements that are used in this thesis. These definitions are informal but meant to be sufficient to read this thesis. Interested readers are welcome to read more about these notions from Clarkson, Chong and Myers [2], Juels, Catalano and Jakobsson [9], Chondros, Delis, and Gavatha et al. [5] and Kiayias, Zacharias and Zhang [11].

Electronic voting system The casting and counting of votes in a election require information and communication technologies.

Supervised voting system The voter votes at a polling station.

Remote voting system The voter does not need to go to a polling station to vote.

We concentrate on three security requirements; verifiability, availability and privacy. We present here a general definition for each of these.

Verifiability A voting system is verifiable if it fulfills one or more of the following definitions.

- **Voter Verifiability** The voter can check that her own vote is included in the tally.
- **End-to-end Verifiability** It can be checked that all votes cast are counted, that only authorized votes are counted, and that no votes are changed during counting.

Availability A voting system is available if it is both accessible and understandable.

- **Accessible** The voter is able to vote without any restriction.
- **Understandable** The voter can understand and use the system.

Privacy A voting system is private if it is coercion resistant and if it ensures anonymity.

- **Coercion Resistant** A coercer can not be certain whether the voter cooperated with him, even if they interact while she votes.
- **Anonymity** The identity of the voter and her vote can not be linked.

There is a related notion called receipt-freeness, which is a weaker version of coercion resistance.

Receipt-freeness The voter does not receive a receipt that can prove how she voted.
We remark that a system that is coercion-resistant is consequently receipt-free.

2.2 Mathematics

To ease the notation, we define $[1, n]$ to be $\{1, 2, ..., n\}$.

2.2.1 Public Key Encryption

In public key encryption [17], a message is encrypted with a public key. To decrypt this message, the corresponding private key is needed. We define public key encryption with the following notation in this thesis.

Public Key Encryption

A public cryptosystem is defined by $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ where,

- $\mathcal{P}$ denotes the set of plaintexts.
- $\mathcal{C}$ denotes the set of ciphertexts.
- $\mathcal{K}$ denotes a key generator with output (pk, sk), where pk is the public key and sk is the corresponding secret key.
- $\mathcal{E}$ denotes the set of encryption algorithms.
- $\mathcal{D}$ denotes the set of decryption algorithms.

For any (pk, sk) generated by $\mathcal{K}$ there is an $E_{pk} \in \mathcal{E}$ where $E_{pk}\colon \mathcal{P} \to \mathcal{C}$ and a corresponding $D_{sk} \in \mathcal{D}$, where $D_{sk}\colon \mathcal{C} \to \mathcal{P}$. For every $m \in \mathcal{P}$,

$$D_{sk}(E_{pk}(m)) = m$$

2.2.2 Exponential ElGamal

The exponential ElGamal is a modified version of the ElGamal cryptosystem [17]. A generator g of prime order is raised to the power of the message m, so that g^m is encrypted. While the original ElGamal is homomorphic over multiplication, this system satisfies the additive homomorphic property, $g^{m_1} \cdot g^{m_2} = g^{m_1+m_2}$. After decrypting the ciphertext, g^m is obtained. Because of the discrete logarithm problem, the message m has to be small so that it can be retrieved by known algorithms such as Shanks Algorithm or with a precomputed table. The exponential ElGamal described below can be used on a general group. To ease in reference we use the group $\mathbb{Z}_p^*$ since it is this specific version of exponential ElGamal we refer to later in this thesis.

Exponential ElGamal Public-Key cryptosystem in $\mathbb{Z}_p^*$

Let $\mathbb{G}$ be a subgroup of $\mathbb{Z}_p^*$ with order q and generator g, where both p and q are primes. Assume computing discrete logarithms in $\mathbb{G}$ is infeasible. $\mathbb{G}$ is isomorphic to $\mathbb{Z}_q$.

- $\mathcal{P}= \mathbb{Z}_q$
- $\mathcal{C}= \mathbb{Z}_p^* \times \mathbb{Z}_p^*$
- $(pk, sk) \leftarrow \mathcal{K}$, where $pk = (p, g, \beta)$ and $sk = e$.
  $\beta \equiv g^e \bmod p$.

Let $k, m \in \mathcal{P}$, where k is a secret random number and m is the message to encrypt. We define the encryption algorithm, E_{pk}:

$$E_{pk}(m') = (c_1, c_2),$$

where m'=g^m, $c_1 = g^k \bmod p$, $c_2 = g^m \beta^k \bmod p$.

For (c_1,c_2) $\in \mathcal{C}$, we define the decryption algorithm D_{sk}:

$$D_{sk}(c_1, c_2) = c_2({c_1}^e)^{-1} = g^m \beta^k (g^{ke})^{-1} = g^m g^{ek} (g^{ke})^{-1} = g^m = m'.$$

Then retrieving m from m' either by known algorithms or from a precomputed table.

We note that using exponential El Gamal is not problematic in a voting system since the plaintexts often are candidates from a small set, so that the discrete logarithm may be found.

2.2.3 Negligible function

One of the main goals of using a cryptographic scheme is that no adversary can break the scheme without knowing the key. However, an adversary given unbounded computational power can find the key to the scheme by brute force. Most cryptographic systems in use today assume an adversary with bounded computational power.

Negligible function

The function $neg : \mathbb{N} \to [0, 1]$ is defined as negligible if for all $c > 0$, there exists an $n > N_c$ such that,

$$neg(n) < \frac{1}{n^c}.$$

In a cryptographic system $n \in \mathbb{N}$ is the key length. We observe that, a negligible function multiplied by any polynomial $p(n)$ is still negligible, $neg(n) < \frac{1}{n^c} \cdot p(n)$. We often refer to negligible functions when defining the probability that an adversary breaks the system. If the probability of breaking the scheme is negligible, the probability stays negligible if it is repeated a polynomial number of times. For all $c > 0$, there exists an $n > N_c$ such that,

$$Pr[\text{an adversary breaks the scheme}] \leq \frac{1}{n^c} \cdot p(n).$$

2.2.4 Decisional Diffie Hellman Assumption

Decisional Diffie Hellman Assumption

Let $\mathbb{G}$ be a group generated by g. If the Decisional Diffie Hellman Assumption holds, it is infeasible to distinguish between,

$$\{(g^a, g^b, c) \mid a, b, c \in \mathbb{G}\}$$

and

$$\{(g^a, g^b, g^{ab}) \mid a, b \in \mathbb{G}\}.$$

The Decisional Diffie Hellman Assumption is stronger than assuming that computing the discrete logarithms in the group is infeasible. By knowing a or b one can compute g^{ab} and distinguish c' from c.

2.2.5 Commitment Scheme

A commitment scheme [1] is a method of committing to a value while keeping it secret to others. By this, the scheme has two properties, namely hiding and binding. The commitment gives no information about the value committed, providing the hiding property. The binding property follows because the commitment eliminate the possibility of changing the original value. To open the commitment, a secret opening value has to be revealed by the agent who preformed the commitment. Below we present a general commitment scheme.

Commitment Scheme

Let *ck* the commitment key. A commitment scheme is based on the following:

- For any $m \in \mathcal{P}$
$$Com_{ck}(m) = (c, d)$$
is the commitment/opening pair form of m, where c is the commitment value and d is the opening value.
- The opening of the commitment is presented as
$$Open_{ck}(c, d) = m \in \mathcal{P} \cup \{\perp\},$$
where $\perp$ is returned if the commitment/opening pair (c, d) does not open to any valid message in $\mathcal{P}$.

Below we show how a commitment scheme is used in practice. We assume Bob wants to commit a value m to Alice.

1. Bob generates $Com_{ck}(m) = (c, d)$, and sends c to Alice.
2. To open the commitment, Bob sends d to Alice.
3. Alice computes $Open_{ck}(c, d) = m$ and accepts the value, provided $m \neq \perp$.

In voting systems, commitment schemes can be used to ensure verifiability and privacy of a cast vote.

2.2.6 Threshold Scheme

A threshold scheme [17] is a method of sharing a secret key by dividing the information among several participants. It is called a (k, n)-threshold scheme if n is the number of participants, from which any subgroup of k participants can compute the secret key, but no group of $k - 1$ participants can obtain this information.

Shamir (k, n)-Threshold Scheme

Key Distribution
Let q be a prime. A third party D wants to divide a secret key $a_0 \in \mathbb{Z}_q$ among n participants.
D chooses randomly $k - 1$ elements of $\mathbb{Z}_q$, denoted $a_1, a_2, ..., a_{k-1}$ in secret.
D constructs the polynomial:

$$p(x) = a_0 + a_1 x + a_2 x + ... + a_{k-1} x^{k-1} \mod q$$

Then D chooses n non-zero elements of $\mathbb{Z}_p$, denoted x_i, for $i \in [1, n]$.
Finally, D computes

$$p(x_i) \qquad \text{for } i \in [1, n]$$

and distributes $p(x_i)$ to participant i, for $i \in [1, n]$.

Retrieving Secret Key
Only k honest participants are needed to retrieve the polynomial $p(x)$, and hence the secret key a_0. Given a set of k points, $(x_1, p(x_1)), (x_2, p(x_2)), ..., (x_k, p(x_k))$, the interpolation polynomial in the Lagrange form is the linear combination:

$$L(x) = p(x_1)l_1(x) + p(x_2)l_2(x) + ... + p(x_k)l_k(x) \mod q$$

$$\text{where,} \quad l_j(x) = \prod_{m=1, m \neq j}^{k} \frac{x - x_m}{x_j - x_m}$$

The k participants can then computes $L(0) = a_0$ in order to obtain the secret key.

The threshold scheme described above is used in voting systems to share the secret decryption key of the votes into several independent parties. This supports the verifiability requirement of a voting system, both making the whole system more robust against attacks and making it harder for malicious agents who decrypt, to manipulate votes without being caught.

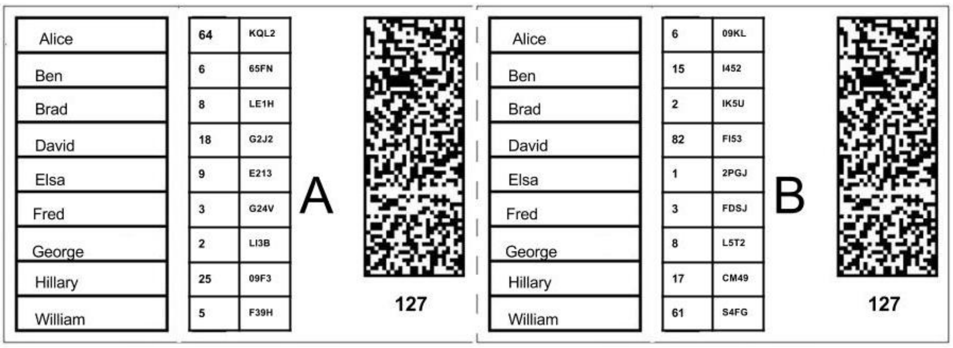
Chapter 3

Demos

3.1 Introduction

Demos [8, 11] is presented as a paper-based electronic voting system that supports verifiability and voter privacy. It has been tested in a pilot experiment during the European elections 2014 in Athens, Greece. It can be used as a supervised or a remote voting system. We present the supervised version.

At the polling station, the voter receives one ballot including left-hand side A and right-hand side B, both containing the candidate list in alphabetical order. For each candidate there is a corresponding vote-code and a code-receipt. They are both randomly chosen for each side, and they are unique for each candidate. The voter must separate side A and side B and choose randomly between them: one side is used for voting and the other one for verification.



Alice	64	KQL2	A
Ben	6	65FN	
Brad	8	LE1H	
David	18	G2J2	
Elsa	9	E213	
Fred	3	G24V	
George	2	LI3B	
Hillary	25	09F3	
William	5	F39H	127

Alice	6	09KL	B
Ben	15	I452	
Brad	2	IK5U	
David	82	FI53	
Elsa	1	2PGJ	
Fred	3	FDSJ	
George	8	L5T2	
Hillary	17	CM49	
William	61	S4FG	127

A ballot including left-hand side A and right-hand side B, having ballot number 127.

Without loss of generality, we assume that she chooses side A for voting and side B for verification. In order to cast her vote, she first scans the QR-code on side A in the voting machine. The names of the candidates appear on the screen. She selects the candidate of her choice and the corresponding code-receipt will appear. She can immediately verify that the code-receipt on the screen corresponds to the code-receipt next to her candidate on the paper-ballot. She writes down the vote-code corresponding to the candidate of her choice from side A and keeps side B for verification. Side A must be destroyed, so no one can figure out which candidate corresponds to her vote-code.

In advance, the election authority in charge of the election has committed to all the information on side A and on side B, so that these can not be changed during the election. The voter can verify that her vote was counted correctly by logging in with her ballot number on the election website. There, all the vote-codes from side A appear in a random order, and one of the vote-codes is marked. To provide coercion-resistance, the names of the candidates do not appear on the screen. Because of that, she can not be sure that this marked vote-code corresponds to the correct candidate.

To verify the system, the voter does two things. First, she checks that the vote-code marked on the screen is the same as she has written down from side A. Moreover, because the voter chooses side A to vote with, the commitment on side B will be opened and available on the website. She verifies that the side B on the screen is exactly the same as the side B she has kept.

The key idea of the system is that the vote-codes of the candidates are different from ballot to ballot. This makes it impossible for a third party to guess which candidate corresponds to the vote-code written on the website. This contributes to the privacy of the system. Furthermore, each voter verifies the correctness of a random side of their ballot on the election website. If the information on the side appearing on the screen is correct, so should the information on the other side. This gives assurance that the ballots are constructed in a correct way contributing to the verifiability of the system.

3.2 Cryptographic description

We present in this section the cryptographic description of Demos. The voting system is based on commitments made before the election of both the vote-codes and the candidates. Most of the work of the election authority is done and published in advance in a committed form, so that they minimize the work after the voting phase. This gives assurance to voters and candidates that the election authority does not cheat during the process.

The code-receipt is not included in the article presenting the mathematical description of the system. When the voter cast her ballot by submitting her vote-code, the corresponding code-receipt appears on the screen. This assures the voter she votes correctly, since the code-receipts are unique. The code-receipt does not seem to have other purposes and is therefore omitted in this thesis.

The bulletin board often takes form of an election website in a voting system. From now on, we use the more technical notion bulletin board instead of election website.

Additionally, the system is presented for an election where the voter votes for one candidate only. The system may also be implemented for an election where the voter can vote for multiple candidates.

3.2.1 Setup phase

There are three different types of agents involved in the voting system: the election authority, the bulletin board and the voters. Demos stresses that the election authority can be split into different parties.

- The election authority generates the setup phase, the ballots and their commitments, tallies the votes and publishes proofs of his honest work.
- The bulletin board passively provides storage of information used for verification and results.
- The voter votes for the candidate of her choice, and can verify that her vote was counted correctly.

First, the election authority includes the set of voters, $\mathcal{V} = \{V_1, V_2, ..., V_l\}$, and the set of candidates, $\mathcal{T} = \{T_1, T_2, ..., T_m\}$ in the voting system. Then, the election authority generates the commitment key, ck, for the commitment scheme.

Commitment Key Generation

The commitment key generation in Demos is based on a group over an elliptic curve. The elliptic curve domain parameters are pk:=(p, a, b, g, q), where

- p is a prime specifying the field $\mathbb{F}_p$.
- $a, b \in \mathbb{F}_p$ define the elliptic curve $E(\mathbb{F}_p)$ by the equation, $E : y^2 = x^3 + ax + b \mod p$.
- $g = (x_0, y_0)$ is an element in $E(\mathbb{F}_p)$ with prime order q.
- $\mathbb{G}$ is the group generated by g and is isomorphic to $\mathbb{Z}_q$.
- it is assumed the Decisional Diffie Hellman Assumption holds over $\mathbb{G}$.

Let ω be a random element in $\mathbb{Z}_q$ and $h = g^\omega$.
The commitment key is $ck = (pk, h)$.

Encoded candidate The election authority encodes the candidates in a number system to facilitate the tallying. In the number system with base $l + 1$, where l is the number of voters and m is the number of candidates, the encoding of candidates is denoted by

$$T_i \leftarrow (l + 1)^{i-1}, \quad i \in [1, m].$$

Thus, the pairs $(T_1, (l + 1)^0), (T_2, (l + 1)^1), ..., (T_m, (l + 1)^{m-1})$ are obtained. Note that the encoding of candidates is the same for all the ballots.

Generating a ballot We now describe the process of generating a ballot. A ballot must include a unique ballot number, two sides A and B containing the list of the candidates in alphabetical order $(T_1, T_2, ..., T_m)$ with their corresponding vote-codes $(C_1, C_2, ..., C_m)$ and QR-codes containing this information.

Generating a ballot

A ballot consists of side A, s^A, and side B, s^B. To create a ballot the election authority does the following:

1. Selects a unique ballot number, denoted *BN*.
2. Selects random vote-codes, $C_i^A \in \mathbb{Z}_q$ for $i \in [1, m]$, unique for each candidate, $T_i \in \mathcal{T}$, on side A. Similarly, selects random $C_i^B \in \mathbb{Z}_q$ on side B.
3. Generates the ballot, $s = (BN, s^A, s^B)$, where $s^A = \{(T_i, C_i^A)\}_{i\in[1,m]}$ and $s^B = \{(T_i, C_i^B)\}_{i\in[1,m]}$.

Commitment scheme To ensure privacy and verifiability, the encoded candidates together with the vote-codes must be kept secret and unchanged. Demos uses therefore a commitment scheme during the setup phase which commits the election authority to the encoding and the vote-codes while keeping these secret to others during the election. The commitments are opened during the tallying phase. As wanted, the commitment scheme gives an assurance that both the encoding of the candidates and the vote-codes remain secret and unchanged. The election authority implements a commitment scheme based on the discrete logarithm problem using the commitment key ck.

Commitment Scheme based on exponential ElGamal

Let $ck = (pk, h)$ be the commitment key defined above. For $m, t \in \mathbb{Z}_q$ where t is random:

$$Com_{ck}(m) = (c, d) = ((g^t, g^m \cdot h^t), t)$$

is the commitment/opening pair form of m.
The commitment/opening pair is homomorphic under operation:

$$Com_{ck}(m_1) \cdot Com_{ck}(m_2) = Com_{ck}(m_1 + m_2)$$

For $m_1, m_2 \in \mathcal{P}$.

Vote-code commitment The election authority generates random $t_i^A, t_i^B \in \mathbb{Z}_q$ and computes the vote-code commitment/opening pairs for $i \in [1, m]$:

$$Com_{ck}(C_i^A) = (c(C_i^A), d(C_i^A)) = ((g^{t_i^A}, g^{C_i^A} \cdot h^{t_i^A}), t_i^A)$$

$$Com_{ck}(C_i^B) = (c(C_i^B), d(C_i^B)) = ((g^{t_i^B}, g^{C_i^B} \cdot h^{t_i^B}), t_i^B)$$

Encoded candidate commitment The election authority generates random $r_i^A, r_i^B \in \mathbb{Z}_q$ and computes the encoded candidate commitment/opening pairs for $i \in [1, m]$:

$$Com_{ck}((l+1)^{i-1}) = (c((l+1)^{i-1}), d((l+1)^{i-1})) = ((g^{r_i^A}, g^{(l+1)^{i-1}} \cdot h^{r_i^A}), r_i^A)$$

$$Com_{ck}((l+1)^{i-1}) = (c((l+1)^{i-1}), d((l+1)^{i-1})) = ((g^{r_i^B}, g^{(l+1)^{i-1}} \cdot h^{r_i^B}), r_i^B)$$

Random permutation of the order of the candidates To support privacy of the vote on the bulletin board, the election authority selects random permutations which shuffle the order of the vote-codes on side A and side B for each ballot. Indeed, if the vote-codes are not shuffled, the position of the marked vote-code on the bulletin board reveals the vote. Without information of the permutation, the privacy of the voter is kept during this phase. For each ballot, the random permutations on side A and side B are:

$$\pi^A : [1, m] \to [1, m]$$

$$\pi^B : [1, m] \to [1, m]$$

The new position of the ith vote-code is denoted by $\pi^A(i)$ and $\pi^B(i)$, respectively.

Finally, the election authority publishes all the information needed on the bulletin board. The opening $(d(C_{\pi(i)}), d((l+1)^{i-1}))$ for $\pi(i) \in [1, m]$ is kept secret.

Published on the bulletin board

- The set of candidates $\mathcal{T}$.
- The set of voters $\mathcal{V}$.
- The commitment key, ck.
- For side A and side B of each ballot; the ballot number, *BN*, and the pair of vote-code/encoded candidate in committed form:

$$(c(C_{\pi(i)}), c((l+1)^{i-1})) \quad \text{for}\, \pi(i) \in [1, m]$$

  in the permutation order, π^A, π^B, respectively.

3.2.2 Voting phase

During the voting phase, the voter chooses randomly between side A and side B by tossing a coin. Without loss of generality, let's assume she votes with side A. She chooses the candidate $T_i \in \mathcal{T}$ of her choice, by selecting the corresponding vote-code C_i^A. The vote cast is $V_{\text{cast}} = (BN, s^A, C_i^A)$ and the receipt obtained is $V_{\text{receipt}} = (BN, s^B, C_i^A)$.

3.2.3 Tallying phase

After the vote is cast, it must be recorded in the system. The other side can be published together with its secret information for verification on the bulletin board.

Retrieving information from a vote

Without loss of generality, we assume the vote cast is $V_{\text{cast}} = (BN, s^A, C_i^A)$.
Then the election authority follows the procedure:

1. Publishes (V_{cast}, s^B) to the bulletin board and opens the commitments of side B.
2. Matches the vote-code C_i^A with the permuted vote-code: $C_{\pi(i)}^A \leftarrow C_i^A$.
3. Marks $C_{\pi(i)}^A$ as voted and sends the corresponding commitment $c((l+1)^{i-1})$ to tallying.

Now, the election authority has sent all the votes in their encoded commitment form to the tally. Due to their homomorphic property, it is the commitments of the encoded candidates that are counted under the tallying.

Tallying of the ballots

We denote C the product of all the encoded candidates commitments for all the votes cast l. That is,

$$C = \prod_{j=1}^{l} c((l+1)^{i_j-1}) = c((l+1)^{i_1-1} + (l+1)^{i_2-1} + ... + (l+1)^{i_l-1}),$$

where i_j corresponds to the candidate choice of voter V_j. The election authority publishes C on the bulletin board with its corresponding opening. We set $Open_{ck}(C) = T$. The election result, R, is calculated in the number system with base $l+1$ by the following algorithm. For $i \in [1, m]$:

- $x_i = T \bmod (l+1)$
- $T = (T - x_i)/(l+1)$
- Return $R = (x_1, x_2, ..., x_m)$

After tallying the ballots, the election authority publishes the result $R = (x_1, x_2, ..., x_m)$, where x_i is the number of votes for the candidate T_i.

3.3 Example

We make a simplified example of the system for a better understanding.

Setup phase The election authority includes two voters, V_1 and V_2 and three candidates namely Anna, Marit and Sigurd in the voting system. Additionally, the election authority encodes the candidates. Anna is encoded to $(2+1)^0 = 1$, Marit is encoded to $(2+1)^1 = 3$ and Sigurd is encoded to $(2+1)^2 = 9$.

Generation of ballots

The election authority proceed with the following:

Generation of ballot s_1

1. Selects *BN*$=$ 23.
2. Respectively, for Anna, Marit and Sigurd,
   - Selects random vote-codes $C_1^A = 241$, $C_2^A = 756$ and $C_3^A = 345$ for side A.
   - Selects random vote-codes $C_1^B = 123$, $C_2^B = 385$ and $C_3^B = 946$ for side B.
3. Generates the ballot $s_1 = (23, s^A, s^B)$, where
   - $s^A = ((\text{Anna}, 241), (\text{Marit}, 756), (\text{Sigurd}, 345))$
   - $s^B = ((\text{Anna}, 123), (\text{Marit}, 385), (\text{Sigurd}, 946))$

Generation of ballot s_2

1. Selects *BN*$=$ 84.
2. Respectively, for Anna, Marit and Sigurd,
   - Selects vote-codes $C_1^A = 256$, $C_2^A = 486$ and $C_3^A = 542$ for side A.
   - Selects vote-codes $C_1^B = 383$, $C_2^B = 430$ and $C_3^B = 639$ for side B.
3. Generates the ballot $s_2 = (84, s^A, s^B)$, where
   - $s^A = ((\text{Anna}, 256), (\text{Marit}, 486), (\text{Sigurd}, 542))$
   - $s^B = ((\text{Anna}, 383), (\text{Marit}, 430), (\text{Sigurd}, 639))$

Then, the election authority selects random permutations to shuffle the vote-codes on each ballot,

Ballot number 23: $\pi_1^A : (1,2,3) \rightarrow (3,2,1)$ and $\pi_1^B : (1,2,3) \rightarrow (1,2,3)$

Ballot number 84: $\pi_2^A : (1,2,3) \rightarrow (2,1,3)$ and $\pi_2^B : (1,2,3) \rightarrow (3,2,1)$

The vote-code and encoding for each side of each ballot are paired up. Finally, the election authority generates the commitment/opening pair of the vote-codes and the encodings of the candidates. The commitments in their permutation order are posted on the bulletin board while the corresponding openings are kept secret by the election authority.

Published on the bulletin board

- The set of candidates: Anna, Marit, Sigurd.
- The set of voters: V_1 and V_2.
- The commitment key, ck.

Ballot number 23:		**Ballot number 84**:	
Side A:	Side B:	Side A:	Side B:
$(c(345), c(9))$	$(c(123), c(1))$	$(c(486), c(3))$	$(c(639), c(9))$
$(c(756), c(3))$	$(c(385), c(3))$	$(c(256), c(1))$	$(c(430), c(3))$
$(c(241), c(1))$	$(c(946), c(9))$	$(c(542), c(9))$	$(c(383), c(1))$

Voting phase After these computations by the election authority, the voting phase can start.

- V_1 picks the ballot $s_1 = (23, s^A, s^B)$, flips a coin, and votes with side A, s^A, for Anna.
    1. The vote cast is $V_{\text{cast}} = (23, s^A, 241)$.
    2. The vote receipt received is $V_{\text{receipt}} = (23, s^B, 241)$.
- V_2 picks the ballot $s_2 = (84, s^A, s^B)$, flips a coin, and votes with side B, s^B, for Sigurd.
    1. The vote cast is $V_{\text{cast}} = (38, s^B, 639)$.
    2. The vote receipt received is $V_{\text{receipt}} = (38, s^A, 639)$.

Tallying phase After the voting is done, the election authority retrieves information of the vote.

Retrieving information from the votes

The election authority:

1. Publishes $V_{\text{cast}} = (23, s^A, 241)$ and $V_{\text{cast}} = (38, s^B, 639)$ on the bulletin board.
2. Opens the commitments of side B for ballot 23 and side A for ballot 38 and the commitments of the vote-code cast, 241 and 639.
3. Matches the vote-codes with their corresponding encoded candidate commitments $(241, c(1))$ and $(639, c(9))$ and mark them as voted.
4. Sends the encoded candidate commitments $c(1)$ and $c(9)$ to tallying.

Now the election authority tallies the votes.

Tallying of votes

The tallying is done homomorphically by computing the product of the commitment/opening pair,

$$Com_{ck}(1) \cdot Com_{ck}(9) = Com_{ck}(1+9) = Com_{ck}(10) = (c(10), d(10))$$

The election authority publishes c(10) along with its opening, d(10), on the bulletin board, hence $T = Open_{ck}(c(10), d(10)) = 10$ is now known.

The election result, R is calculated in the number system with base $l + 1 = 3$ by the algorithm presented earlier:

- $x_1 = T \text{ mod } l + 1 = 10 \text{ mod } 3 = 1$
- $T = (T - x_1)/(l + 1) = 9/3 = 3$
- $x_2 = T \text{ mod } l + 1 = 3 \text{ mod } 3 = 0$
- $T = (T - x_2)/(l + 1) = 3/3 = 1$
- $x_3 = T \text{ mod } l + 1 = 1 \text{ mod } 3 = 1$

$\Rightarrow R = (x_1, x_2, x_3) = (1, 0, 1)$.

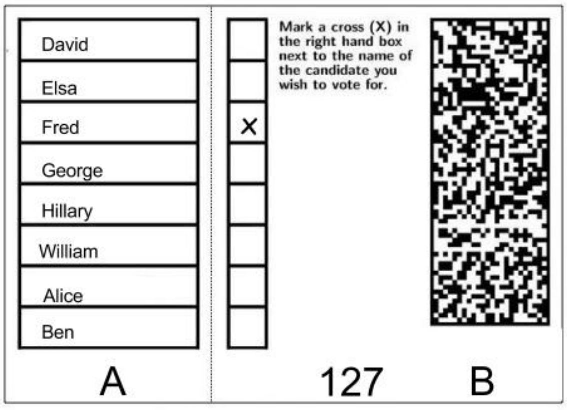
The result states that Anna got one vote, Marit got zero votes and Sigurd got one vote.

Chapter 4

Prêt-à-voter

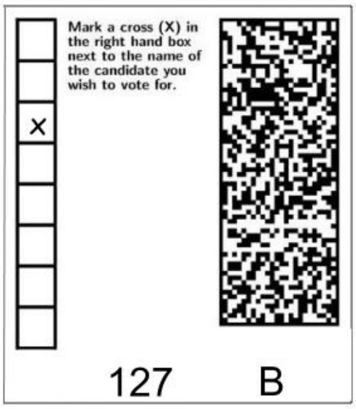
4.1 Introduction

Prêt-à-voter is a supervised electronic voting system based on paper ballots. At the polling station, the voter receives a ballot with two sides, for simplicity called side A and side B. Side A contains the list of the candidates, written in an alphabetical order with a cyclic shift, for each ballot. The voter marks a cross next to the candidate of her choice on side B. Then, she has to divide the two parts.


David
Elsa
Fred
George
Hillary
William
Alice
Ben
A
X
Mark a cross (X) in the right hand box next to the name of the candidate you wish to vote for.
127
B

A completed ballot form with side A and side B having ballot number 127.

Side A must be destroyed at once, so no one knows the order of the candidate list on her ballot. Side B, however, is signed and scanned to be counted, then it is kept by the voter as a receipt.


Mark a cross (X) in the right hand box next to the name of the candidate you wish to vote for.
X
127
B

Side B of the ballot form, scanned to be counted and kept as receipt.

Side B contains a cross marking a position but not a candidate. She can later verify that her vote was counted correctly by entering her ballot number on the election website. If nothing went wrong, her exact side B should appear on the website. The key idea of the system is that the candidate list is random, varying from ballot to ballot. This makes it impossible for a third party to guess which exact candidate order corresponds to the receipt, side B.

Prêt-à-voter is a family of voting systems [13, 15, 4, 3, 14]. We concentrate in this thesis on the article summarising Prêt-à-voter from 2010 [14], presenting two different ways of designing the ballot forms, and how these can be tallied. First, we present the design based on decryption mix-net used when the ballots are pre-printed. Then, we present re-encryption mix-net, used when the ballots are printed on-demand.

4.2 Cryptographic description by decryption mix-net

The ballot is constructed in a way so that the encrypted vote, side B with the marked cross, can be decrypted. Simply explained, the QR-code contains encrypted information of which box corresponds to which candidate. So that when the vote is cast, it is possible to retrieve which candidate the marked cross corresponds to.

4.2.1 Setup phase

There are four different types of agents involved in the decryption mix-net version of Prêt-à-voter: the election authority, the mix-servers, the bulletin board and the voters.

- The election authority includes the set of voters, the set of candidates and the set of mix-servers in the voting system and designs the ballots by using public keys generated by the mix-servers.

- The mix-servers generate key-pairs both for encryption of ballots and decryption of votes and shuffle all the ballots.

- The bulletin board passively provides storage of information used for verification and results.
- The voter votes for the candidate of her choice, and can verify that her vote was counted correctly.

First, the election authority includes the set of voters, $\mathcal{V} = \{V_1, V_2, ..., V_l\}$, the set of candidates, $\mathcal{T} = \{T_1, T_2, ..., T_m\}$ and the set of mix-servers $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ in the voting system. Each mix-server $S_i \in \mathcal{S}$ generates two pairs of keys, $(pk_{i,1}, sk_{i,1})$ and $(pk_{i,2}, sk_{i,2})$ using RSA key generation [17].

Key Generation using RSA

A key pair, (pk, sk), is generated using *RSA*. The mix-server proceeds with the following:

1. Generates two large primes, p and q, such that $p \neq q$ and differ in length by a few digits.
2. Calculates $n = pq$ and $\phi(n) = (p-1)(q-1)$, where ϕ is the Eulers' totient function.
3. Chooses a random $b \in [1, \phi(n)]$ such that $gcd(b, \phi(n)) = 1$.
4. Calculates $a = b^{-1} \bmod \phi(n)$.

The key pair (pk, sk) is now generated where, $pk = (n, b)$ and $sk = (p, q, a)$.

To prepare against failing mix-servers, the keys are shared among all the mix-servers in a threshold scheme presented in section 2.2.6.

Encryption We denote the alphabetical ordered candidate list $(T_1, T_2, ..., T_m)$. First the election authority chooses a random shift $\alpha_{0,2}$ of this list, denoted $(T_1, T_2, ..., T_m)_{\alpha_{0,2} \bmod m}$ which is the order used during the tallying of the votes.

Assuming there are k mix-servers, the election authority continues by selecting $2k$ random seed values $\{t_{1,1}, t_{1,2}, t_{2,1}, t_{2,2}, ..., t_{k,1}, t_{k,2}\}$. With a hash of the seed $t_{1,1}$, denoted $h_{1,1}$, the election authority shifts the candidate list $(T_1, T_2, ..., T_m)_{\alpha_{0,2} \bmod m}$ to obtain $(T_1, T_2, ..., T_m)_{\alpha_{0,2}+h_{1,1} \bmod m}$. He repeats this procedure, and the candidate list printed on the ballot is $(T_1, T_2, ..., T_m)_{\alpha_{0,2}+\Sigma_{i=1}^{k}(h_{i,1}+h_{i,2}) \bmod m} = (T_1, T_2, ..., T_m)_{\alpha_{0,2+h \bmod m}}$.

To not reveal how the candidate list has been shifted, the election authority encrypts $\alpha_{0,2}$ and the random seed values $\{t_{1,1}, t_{1,2}, t_{2,1}, t_{2,2}, ..., t_{k,1}, t_{k,2}\}$ layers by layers into an onion using the $2k$ public keys generated by the mix-servers. How this is done, is explained thoroughly later. In this design any public cryptosystem can be implemented, but we present a case using the *RSA public cryptosystem.*

RSA public cryptosystem

Let $n = pq$ where, p and q are primes.

1. $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$.
2. $\mathcal{K} = (pk, sk) = ((n, b), (p, q, a))$ where, $ab \equiv 1 \bmod \phi(n)$.
3. The encryption algorithm $E_{pk} \in \mathcal{E}$ is,
$$E_{pk}(m) = m^b \bmod n.$$
4. The decryption algorithm $D_{sk} \in \mathcal{D}$ is,
$$D_{sk}(c) = c^a \bmod n.$$

For each ballot the election authority does the following:

1. Selects randomly $\alpha_{0,2} \in \mathbb{Z}$, which modulo m represents the cyclic shift of a candidate list, $(T_1, T_2, ..., T_m)_{\alpha_{0,2} \bmod m}$.
2. Selects randomly $t_{i,1} \in \mathbb{Z}_{n_{i,1}}$ and $t_{i,2} \in \mathbb{Z}_{n_{i,2}}$ for $i \in [1, k]$:
   (a) Shifts the candidate list by a hash of $t_{i,1}$, $h_{i,1} = H(t_{i,1}) \bmod m$ and $t_{i,2}$, $h_{i,2} = H(t_{i,2}) \bmod m$ such that,
   $$(T_1, T_2, ..., T_m)_{\alpha_{0,2} + \Sigma_{i=1}^{k}(h_{i,1} + h_{i,2}) \bmod m}.$$
   (b) Encrypts the seeds $t_{i,1}$ of the shift $h_{i,1}$ and $t_{i,2}$ of the shift $h_{i,2}$ with the public key $pk_{i,1} = (n_{i,1}, b_{i,1})$ and $pk_{i,2} = (n_{i,2}, b_{i,2})$ respectively such that,
   $$\alpha_{i,1} = E_{pk_{i,1}}(t_{i,1}, \alpha_{i-1,2}) = (t_{i,1}, \alpha_{i-1,2})^{b_{i,1}} \bmod n_{i,1} = (t_{i,1}^{b_{i,1}}, \alpha_{i-1,2}^{b_{i,1}}) \bmod n_{i,1},$$
   $$\alpha_{i,2} = E_{pk_{i,2}}(t_{i,2}, \alpha_{i,1}) = (t_{i,2}, \alpha_{i,1})^{b_{i,2}} \bmod n_{i,2} = (t_{i,2}^{b_{i,2}}, \alpha_{i,1}^{b_{i,2}}) \bmod n_{i,2}.$$

From the initial list $(T_1, T_2, ..., T_m)_{\alpha_{0,2}}$, the candidate list is cyclically shifted by,
$$h = \Sigma_{i=1}^{k} h_{i,1} + h_{i,2} \bmod m.$$

The onion of the initial order $\alpha_{0,2}$ and the seed value $t_{i,j}$ of the shift $h_{i,j}$ for $i \in [1, k]$ and for $j \in [1, 2]$ is,
$$\alpha = \alpha_{k,2} = E_{pk_{k,2}}(t_{k,2}, E_{pk_{k,1}}(..., E_{pk_{1,2}}(t_{1,2}, E_{pk_{1,1}}(t_{1,1}, \alpha_{0,2})))).$$

Generating a ballot We now describe the process of generating a ballot. A ballot must include two sides, A and B. Side A contains the list of the candidates in an alphabetical order $(T_1, T_2, ..., T_m)$ with a cyclic shift $\alpha_{0,2} + h \bmod m$. Side B contains the ballot number, boxes aligned with the candidate list and the QR-code containing encrypted information.

Generating a ballot

A ballot consists of side A, s^A, and side B, s^B. To create a ballot the election authority does the following:

1. Selects a unique ballot number, denoted *BN*.
2. Selects an initial order $\alpha_{0,2}$ and computes the cyclic shift h of the candidate list $(T_1, T_2, ..., T_m)_{\alpha_{0,2} \bmod m}$, obtaining $(T_1, T_2, ..., T_m)_{\alpha_{0,2}+h \bmod m}$.
3. Encrypts the initial order of the candidate list $\alpha_{0,2}$ together with the seeds $t_{i,j}$ of the shifts $h_{i,j}$ for $i \in [1, k]$ and $j \in [1, 2]$, into the onion α which is encoded into a QR-code.
4. Generates the ballot, $s = (s^A, s^B)$ where, $s^A = (T_1, T_2, ..., T_m)_{\alpha_{0,2}+h \bmod m}$ and $s^B = \{BN, \text{QR}\}$.

4.2.2 Voting phase

During the voting phase, the voter receives a ballot $s = (s^A, s^B)$. She chooses the candidate of her choice, $T_i \in \mathcal{T}$, and marks a cross in the corresponding box. The vote cast is $V_{\text{cast}} = (s^B, \beta)$ where β is the position of the cross. She destroys s^A and receives the receipt, $V_{\text{receipt}} = (s^B, \beta)$. Note that $V_{\text{cast}} = V_{\text{receipt}}$ in Prêt-à-voter.

4.2.3 Tallying phase

Decryption The encrypted vote, V_{cast}, contains the position of the cross and a corresponding onion where the initial order of the candidate list is encrypted (α, β). To decrypt their part, mix-servers use their two secret keys to peel off two layers of the onion. Instead of shifting the order of the list as the election authority did during the encryption, the mix-server shifts the place of the cross. He then passes the partly-decrypted onion and the new cross to the next mix-server who follows the same procedure. The result is a specific position, $\beta_{0,2}$, of the cross, together with a specific order of the candidate list $\alpha_{0,2}$. These are matched to obtain the vote.

We define $\beta = \beta_{k,2}$ to be the position of the cross made by the voter on the ballot. For every ballot, each mix-server $S_i \in \mathcal{S}$ does the following:

1. Retrieves the pair $(\alpha_{i,2}, \beta_{i,2})$.
2. Decrypts $\alpha_{i,2}$ one layer, using his secret key $sk_{i,2}$, $D_{sk_{i,2}}(\alpha_{i,2}) = (t_{i,2}, \alpha_{i,1})$.
3. Computes the hash of $t_{i,2}$, $h_{i,2} = H(t_{i,2}) \bmod m$.
4. Calculates the cyclic shift of the cross, $\beta_{i,1} = \beta_{i,2} - h_{i,2}$.
5. Obtains the pair $(\alpha_{i,1}, \beta_{i,1})$.
6. Decrypts $\alpha_{i,1}$ one layer, using his secret key $sk_{i,1}$, $D_{sk_{i,1}}(\alpha_{i,1}) = (t_{i,1}, \alpha_{i-1,2})$.

7. Computes the hash of $t_{i,1}$, $h_{i,1} = H(t_{i,1}) \bmod m$.

8. Calculates the cyclic shift of the cross, $\beta_{i-1,2} = \beta_{i,1} - h_{i,1}$.

9. Obtains the pair $(\alpha_{i-1,2}, \beta_{i-1,2})$.

After $2k$ decryption, the pair $(\alpha_{0,2}, \beta_{0,2})$ is obtained. The initial order of the candidate list, $(T_1, T_2, ..., T_m)_{\alpha_{0,2} \bmod m}$, can be calculated and matched with the cross in position $\beta_{0,2}$.

Mix-net Above we have described how a mix-server decrypts his part of the onion for one vote. To provide privacy of the votes in the tallying phase, each mix-server must also shuffle the collection of pairs $(\alpha_{i,1}, \beta_{i,1})$ and $(\alpha_{i-1,2}, \beta_{i-1,2})$. Each mix-server $S_i \in \mathcal{S}$ retrieves a collection $L_{i,2} = (B^1_{i,2}, B^2_{i,2}, ..., B^l_{i,2})$ from the bulletin board where, $B^j_{i,2} = (\alpha^j_{i,2}, \beta^j_{i,2})$ is the vote from V_j, for $j \in [1, l]$. He decrypts, and publishes the decrypted pairs in permuted order on the bulletin board. This is done twice for each mix-server, using their two secret keys. The next mix-server continues with the same procedure, decrypting each ballot with his secret keys and shuffling the decrypted collection of pairs.

The mix-server S_i does the following using his secret keys $sk_{i,2}$ and $sk_{i,1}$:

1. Retrieves the collection $L_{i,2} = (B^1_{i,2}, B^2_{i,2}, ..., B^l_{i,2})$ from the bulletin board.

2. Decrypts $D_{sk_{i,2}}(B_{i,2})^j = B^j_{i,1}$ for $j \in [1, l]$.

3. Selects a permutation, π, randomly.

4. Permutes the collection such that, $L_{i,1} = (B^{\pi(1)}_{i,1}, B^{\pi(2)}_{i,1}, ..., B^{\pi(l)}_{i,1})$.

5. Decrypts $D_{sk_{i,1}}(B_{i,1})^j = B^j_{i-1,2}$ for $j \in [1, l]$.

6. Selects a permutation, π, randomly.

7. Permutes the collection such that, $L_{i-1,2} = (B^{\pi(1)}_{i-1,2}, B^{\pi(2)}_{i-1,2}, ..., B^{\pi(l)}_{i-1,2})$.

8. Publishes $L_{i-1,2}$ to the bulletin board.

4.3 Example

We make a simplified example of the system for a better understanding.

Setup phase First, the election authority includes one voter, V, three candidates $\mathcal{T} = \{\text{Anna, Marit, Sigurd}\}$ and two mix-servers, $\mathcal{S} = \{S_1, S_2\}$ in the voting system. Since there are two mix-servers, the election authority generates the ballot with four random values $\{t_{1,1}, t_{1,2}, t_{2,1}, t_{2,2}\}$. The mix-servers, S_1 and S_2, generate two public and two secret keys, denoted $(pk_{1,1}, sk_{1,1}), (pk_{1,2}, sk_{1,2})$ and $(pk_{2,1}, sk_{2,1}), (pk_{2,2}, sk_{2,2})$ respectively.

Encryption The election authority encrypts the ballot of the voter in the following way:

1. Selects randomly $\alpha_{0,2} \in \mathbb{Z} = 2$, which represents the shift of the alphabetical ordered candidate list,

$$(\text{Anna, Marit, Sigurd})_2 = (\text{Marit, Sigurd, Anna}).$$

2. Selects randomly $t_{i,j} \in \mathbb{Z}_{n_{i,j}}$ for $i \in [1,2]$ and $j \in [1,2]$,

   (a) Computes
$$h_{1,1} = H(t_{1,1}) = 2 \bmod 3$$
$$h_{1,2} = H(t_{1,2}) = 2 \bmod 3$$
$$h_{2,1} = H(t_{2,1}) = 1 \bmod 3$$
$$h_{2,2} = H(t_{2,2}) = 0 \bmod 3$$

   (b) Encrypts the seed $t_{i,j}$ of the shift $h_{i,j}$ with the public key $pk_{i,j} = (n_{i,j}, b_{i,j})$ for $i \in [1,2]$ and $j \in [1,2]$ such that,
$$\alpha_{1,1} = E_{pk_{1,1}}(t_{1,1}, \alpha_{0,2})$$
$$\alpha_{1,2} = E_{pk_{1,2}}(t_{1,2}, \alpha_{1,1})$$
$$\alpha_{2,1} = E_{pk_{2,1}}(t_{2,1}, \alpha_{1,2})$$
$$\alpha_{2,2} = E_{pk_{2,2}}(t_{2,2}, \alpha_{2,1})$$

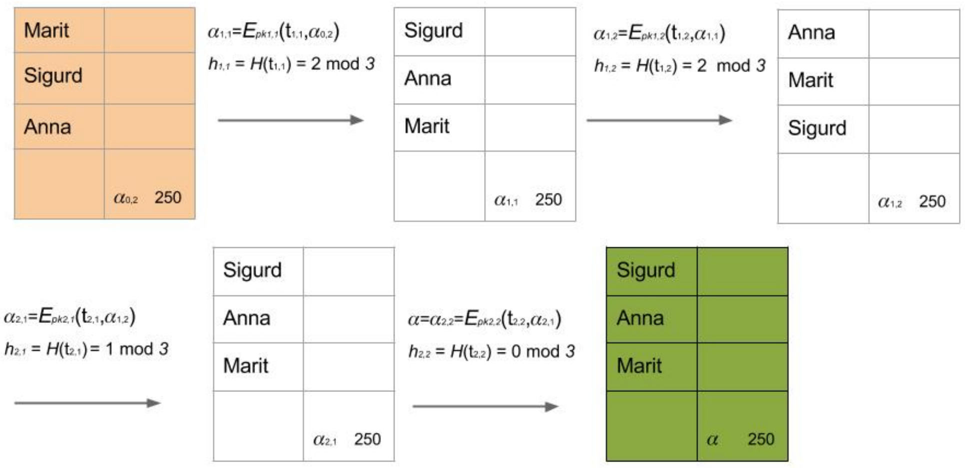
3. The initial list (Marit, Sigurd, Anna) is cyclically shifted by, $h = \Sigma_{i=0}^{3}(h_i) = 5 = 2$ mod 3,
$$(\text{Marit, Sigurd, Anna})_2 = (\text{Sigurd, Anna, Marit}).$$
   The onion of the initial order $\alpha_{0,2}$ and the seed value $t_{i,j}$ of the shift $h_{i,j}$ for $i \in [1,2]$ and $j \in [1,2]$ is,
$$\alpha = \alpha_{2,2} = E_{pk_{2,2}}(t_{2,2}, E_{pk_{2,1}}(t_{2,1}(E_{pk_{1,2}}(t_{1,2}, E_{pk_{1,1}}(t_{1,1}, \alpha_{0,2}))))).$$

Generating a ballot Now the election authority generates the ballot with the encrypted information above.

Generating a ballot

A ballot consists of side A, s^A, and side B, s^B. To create the ballot the election authority does the following:

1. Selects a unique ballot number, denoted *BN*$= 250$.
2. Selects the initial order $\alpha_{0,2} = 2$ and computes the cyclic shift $h = 2$ of the candidate list (Marit, Sigurd, Anna)$_2$, obtaining (Sigurd, Anna, Marit).
3. Encrypts the initial order of the candidate list $\alpha_{0,2} = 2$ together with the seeds $t_{i,j}$ of the shifts $h_{i,j}$ for $i \in [1,2]$ and $j \in [1,2]$, into the onion α which is encoded into a QR-code.
4. Generates the ballot, $s = (s^A, s^B)$ where, s^A =(Sigurd, Anna, Marit) and $s^B = \{250,\text{QR}\}$.


Marit
Sigurd
Anna
$\alpha_{0,2}$ 250
$\alpha_{1,1}=E_{pk1,1}(t_{1,1},\alpha_{0,2})$
$h_{1,1} = H(t_{1,1}) = 2 \bmod 3$
Sigurd
Anna
Marit
$\alpha_{1,1}$ 250
$\alpha_{1,2}=E_{pk1,2}(t_{1,2},\alpha_{1,1})$
$h_{1,2} = H(t_{1,2}) = 2 \bmod 3$
Anna
Marit
Sigurd
$\alpha_{1,2}$ 250
$\alpha_{2,1}=E_{pk2,1}(t_{2,1},\alpha_{1,2})$
$h_{2,1} = H(t_{2,1}) = 1 \bmod 3$
Sigurd
Anna
Marit
$\alpha_{2,1}$ 250
$\alpha=\alpha_{2,2}=E_{pk2,2}(t_{2,2},\alpha_{2,1})$
$h_{2,2} = H(t_{2,2}) = 0 \bmod 3$
Sigurd
Anna
Marit
α 250

Voting phase Now, the voter makes her choice and votes for Anna, obviously. She separates the two parts: side A is thrown away while side B is scanned, and then kept for verification by the voter.

Decryption The two mix-servers decrypt $\alpha = \alpha_{2,2}$ and shift the place of the cross namely $\beta = \beta_{2,2} = 1$ to obtain the vote cast.

Mix-server $S_2 \in \mathcal{S}$ does the following:

1. Retrieves the pair $(\alpha_{2,2}, \beta_{2,2})$.
2. Decrypts $\alpha_{2,2}$ one layer, using his secret key $sk_{2,2}$, $D_{sk_{2,2}}(\alpha_{2,2}) = (t_{2,2}, \alpha_{2,1})$.
3. Computes the hash of $t_{2,2}$, $h_{2,2} = H(t_{2,2}) = 0 \bmod 3$.
4. Calculates the cyclic shift of the cross, $\beta_{2,1} = \beta_{2,2} - h_{2,2} = 1 - 0 = 1 \bmod 3$.
5. Obtains the pair $(\alpha_{2,1}, \beta_{2,1})$.
6. Decrypts $\alpha_{2,1}$ one layer, using his secret key $sk_{2,1}$, $D_{sk_{2,1}}(\alpha_{2,1}) = (t_{2,1}, \alpha_{1,2})$.
7. Computes the hash of $t_{2,1}$, $h_{2,1} = H(t_{2,1}) = 1 \bmod 3$.
8. Calculates the cyclic shift of the cross, $\beta_{1,2} = \beta_{2,1} - h_{2,1} = 0 \bmod 3$.
9. Obtains the pair $(\alpha_{1,2}, \beta_{1,2})$.

Mix-server $S_1 \in \mathcal{S}$ does the following:

1. Retrieves the pair $(\alpha_{1,2}, \beta_{1,2})$.
2. Decrypts $\alpha_{1,2}$ one layer, using his secret key $sk_{1,2}$, $D_{sk_{1,2}}(\alpha_{1,2}) = (t_{1,2}, \alpha_{1,1})$.
3. Computes the hash of $t_{i,2}$, $h_{i,2} = H(t_{i,2}) = 2 \bmod 3$.
4. Calculates the cyclic shift of the cross, $\beta_{1,1} = \beta_{1,2} - h_{1,2} = 0 - 2 = 1 \bmod 3$.

5. Obtains the pair $(\alpha_{1,1}, \beta_{1,1})$.

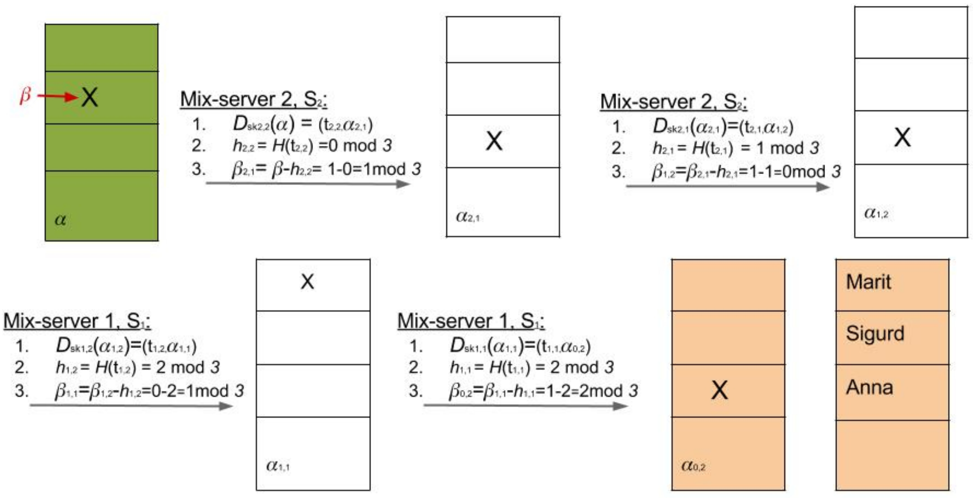
6. Decrypts $\alpha_{1,1}$ one layer, using his secret key $sk_{1,1}$, $D_{sk_{1,1}}(\alpha_{1,1}) = (t_{1,1}, \alpha_{0,2})$.

7. Computes the hash of $t_{1,1}$, $h_{1,1} = H(t_{1,1}) = 2 \bmod 3$.

8. Calculates the cyclic shift of the cross, $\beta_{0,2} = \beta_{1,1} - h_{1,1} = 1 - 2 = 2 \bmod 3$.

9. Obtains the pair $(\alpha_{0,2}, \beta_{0,2})$.

After four decryptions, $\alpha_{0,2} = 2$ is obtained, and the initial order of the candidate list, (Marit, Sigurd, Anna), can be calculated, and matched with the cross which is decrypted to be in position, $\beta_{0,2} = 2$. A vote for Anna is counted.



β
X
α
Mix-server 2, S_2:
1. $D_{sk2,2}(\alpha) = (t_{2,2}, \alpha_{2,1})$
2. $h_{2,2} = H(t_{2,2}) = 0 \bmod 3$
3. $\beta_{2,1} = \beta - h_{2,2} = 1-0=1 \bmod 3$
X
$\alpha_{2,1}$
Mix-server 2, S_2:
1. $D_{sk2,1}(\alpha_{2,1})=(t_{2,1}, \alpha_{1,2})$
2. $h_{2,1} = H(t_{2,1}) = 1 \bmod 3$
3. $\beta_{1,2}=\beta_{2,1}-h_{2,1}=1-1=0 \bmod 3$
X
$\alpha_{1,2}$
Mix-server 1, S_1:
1. $D_{sk1,2}(\alpha_{1,2})=(t_{1,2}, \alpha_{1,1})$
2. $h_{1,2} = H(t_{1,2}) = 2 \bmod 3$
3. $\beta_{1,1}=\beta_{1,2}-h_{1,2}=0-2=1 \bmod 3$
X
$\alpha_{1,1}$
Mix-server 1, S_1:
1. $D_{sk1,1}(\alpha_{1,1})=(t_{1,1}, \alpha_{0,2})$
2. $h_{1,1} = H(t_{1,1}) = 2 \bmod 3$
3. $\beta_{0,2}=\beta_{1,1}-h_{1,1}=1-2=2 \bmod 3$
X
$\alpha_{0,2}$
Marit
Sigurd
Anna

As illustrated the vote is decrypted correctly.

4.4 Cryptographic description by re-encryption mix-net

Re-encryption mix-net is the printed-on-demand version of Prêt-à-voter. The basic idea is that the voter prints her ballot just before voting so that the election authority does not have access to ballots before the voting phase. Because the ballots are printed on-demand, each ballot contains two QR-codes encoding the encryption of the order of the candidate list. The QR-code on side A is decrypted by the ballot printer at once, so that the voter receives a ballot with a candidate list. The QR-code on side B is decrypted by the tellers during the tallying of the votes.

In this description, we omit an example and some parts of the cryptography that are very similar to the decryption mix-net version of Prêt-à-voter and focus more on the difference between these versions.

4.4.1 Setup phase

There are five different types of agents involved in the re-encryption mix-net version of Prêt-à-voter: the election authority divided into groups, the mix-servers, the tellers, the bulletin board and the voters.

- The election authority includes the set of voters, the set of candidates, the set of mix-servers and the set of tellers in the voting system. The election authority divided, into groups, generates the ballot forms.
- The mix-servers shuffle and re-encrypt the received votes.
- The tellers generate key-pairs both for encryption of ballots and decryption of votes and publish the election result.
- The bulletin board passively provides storage of information used for verification and results.
- The voter votes for the candidate of her choice, and can verify that her vote was counted correctly.

For simplicity, we assume that the election authority divided into groups, the mix-servers and the tellers are all divided into k groups. In reality, this number can be different for each types of agents.

The election authority divided into groups, $\mathcal{G} = \{G_1, G_2, .., G_k\}$, determines the set of mix-servers $\mathcal{S} = \{S_1, S_2, ..., S_k\}$, the set of voters $\mathcal{V} = \{V_1, V_2, ..., V_l\}$, the set of candidates $\mathcal{T} = \{T_1, T_2, ..., T_m\}$ and the set of tellers $\mathcal{N} = \{N_1, N_2, ..., N_k\}$. The tellers and the ballot printer generate two independent keys using exponential ElGamal presented in chapter 2.1.2.

Key Generation using exponential ElGamal

We assume the ElGamal parameters (g, p, q) are made public in advance. p and q are large primes such that $q \mid p - 1$, and g is a generator of $\mathbb{Z}_q$ which is isomorphic to a subgroup of $\mathbb{Z}_p^*$ with order q.

- The ballot printer randomly selects a secret key $sk_r \in \mathbb{Z}_q$ and reveals its public key $h_r = g^{sk_r}$.
- The set of tellers $\mathcal{N}$ generates the secret key $sk_t \in \mathbb{Z}_q$ in a threshold fashion. Then publishes the corresponding public key $h_t = g^{sk_t}$.

Encryption The election authority divided into groups, $G_1, G_2, .., G_k$, is in charge of the encryption using the public keys. They have to encrypt the order of the candidate list in two ways. The first encryption, using h_r is decrypted later by the ballot printer during the voting phase. The second encryption using h_t is decrypted later by the set of tellers during the tallying phase.

1. G_1 randomly selects $\alpha_1 \in \mathbb{Z}_m$ and $x_1, y_1 \in \mathbb{Z}_q$ to generate an encryption pair
$$(g^{x_1}, h_r^{x_1} \cdot g^{-\alpha_1}) \text{ and } (g^{y_1}, h_t^{y_1} \cdot g^{-\alpha_1}).$$
2. For $i \in [2, k]$:
   - G_i randomly selects $\alpha'_i \in \mathbb{Z}_m$ and x'_i, $y'_i \in \mathbb{Z}_q$ to generate an intermediate encryption pair
   $$(g^{x'_i}, {h_r}^{x'_i} \cdot g^{-\alpha'_i}) \text{ and } (g^{y'_i}, h_t^{y'_i} \cdot g^{-\alpha'_i}).$$
   - G_i multiplies the intermediate onion pair with the encryption pair received from G_{i-1}
   $$(g^{x_i}, {h_r}^{x_i} \cdot g^{-\alpha_i}) = (g^{x'_i}, {h_r}^{x'_i} \cdot g^{-\alpha'_i}) \cdot (g^{x_{i-1}}, {h_r}^{x_{i-1}} \cdot g^{-\alpha_{i-1}})$$
   $$(g^{y_i}, {h_t}^{y_i} \cdot g^{-\alpha_i}) = (g^{y'_i}, {h_t}^{y'_i} \cdot g^{-\alpha'_i}) \cdot (g^{y_{i-1}}, {h_t}^{y_{i-1}} \cdot g^{-\alpha_{i-1}}).$$
3. The final encryption pair is: $(g^x, {h_r}^x \cdot g^{-\alpha})$ and $(g^y, {h_t}^y \cdot g^{-\alpha})$, where
$$x = x_k = x_1 + \Sigma_{j=2}^k x'_j \mod q$$
$$y = y_k = y_1 + \Sigma_{j=2}^k y'_j \mod q$$
$$\alpha = \alpha_k = \alpha_1 + \Sigma_{j=2}^k \alpha'_j \mod q.$$

The order of the candidate list written on the ballot is denoted by α. Note that α must be equal in both encryption.

Generating a ballot We now describe the process of generating a ballot. A ballot must include two sides, A and B. Side A contains a QR-code containing encrypted information of the list of the candidates in an alphabetical order $(T_1, T_2, ..., T_m)$ with a cyclic shift α. Side B contains the ballot number, boxes aligned with the candidate list and the QR-code containing the same information encrypted in a different way.

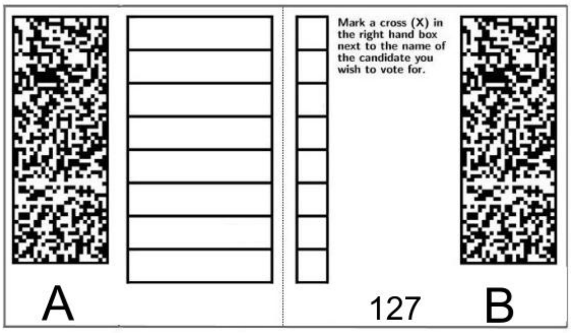
Generating a ballot

A ballot consists of side A, s^A, and side B, s^B. To create a ballot the election authority does the following:

1. Selects a unique ballot number, denoted *BN*.
2. Generates an order α of the candidate list obtaining $(T_1, T_2, ..., T_m)_\alpha$.
3. Encrypts the order of the candidate list α using h_r which is encoded into the QR$_A$-code on side A.
4. Encrypts the order of the candidate list α using h_t which is encoded into the QR$_B$-code on side B.
5. Generates the ballot, $s = (s^A, s^B)$ where, $s^A = \{\text{QR}_A\}$ and $s^B = \{BN, \text{QR}_B\}$.

4.4.2 Voting phase

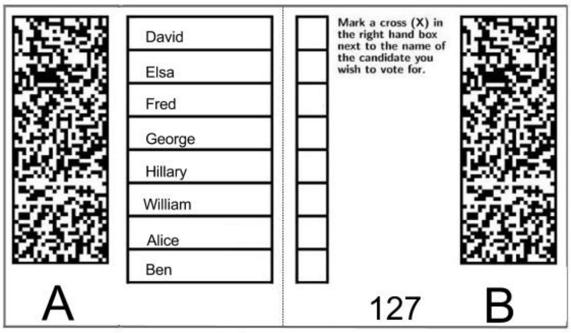
During the voting phase, the voter receives a ballot $s = (s^A, s^B)$.



Mark a cross (X) in the right hand box next to the name of the candidate you wish to vote for.
A
127
B

A ballot before the ballot printer has printed on the candidate list.

She inserts the ballot into the printer. The ballot printer decrypts the code on side A, QR$_A$ by using the secret key sk_r, and prints out the candidate list in the order $(T_1, T_2, ..., T_m)_\alpha$ corresponding to the decrypted code.



David
Elsa
Fred
George
Hillary
William
Alice
Ben
Mark a cross (X) in the right hand box next to the name of the candidate you wish to vote for.
A
127
B

A ballot after the ballot printer has printed on the candidate list.

The voter chooses the candidate of her choice, $T_i \in \mathcal{T}$, and marks a cross in the corresponding box on s^B. The vote cast is $V_{\text{cast}} = (s^B, \beta)$ where β is the position of the cross. She destroys s^A and receives the receipt, $V_{\text{receipt}} = (s^B, \beta)$.

4.4.3 Tallying phase

After the voting phase, side B of all the ballots are published on the bulletin board.

Including the mark on the ballot in the encryption The election authority must include the marks on each ballot in the encrypted code $(g^y, {h_t}^y \cdot g^{-\alpha})$ on side B so that it can be tallied. The position of the cross, denoted by β, is included in the encryption in the following way,

$$(g^y, {h_t}^y \cdot g^{-\alpha} \cdot g^{\beta}) = (g^y, {h_t}^y \cdot g^{-\alpha+\beta}).$$

Mix-net In order to provide privacy, the encrypted votes $(g^y, {h_t}^y \cdot g^{-\alpha+\beta})$ are re-encrypted and shuffled in a mix-net by the mix-servers using permutation functions. This procedure is very similar to the decryption and shuffle phase in the decryption mix-net in chapter 4.2.3. However, instead of decrypting, the mix-servers re-encrypt the votes. These re-encryption do not change $-\alpha + \beta$.

Tallying Finally, the random sequence of re-encrypted votes are decrypted by the tellers using their secret key sk_t. After decryption, the tellers finally perform $-\alpha + \beta = \gamma$. The tellers obtain that the voter voted for candidate T_γ in the alphabetical ordered candidate list $(T_1, T_2, ..., T_m)$. The decrypted votes are then included in the tallying.

4.5 Cryptographic description by re-encryption mix-net

Re-encryption mix-net is the printed-on-demand version of Prêt-à-voter. The basic idea is that the voter prints her ballot just before voting so that the election authority does not have access to ballots before the voting phase. Because the ballots are printed on-demand, each ballot contains two QR-codes encoding the encryption of the order of the candidate list. The QR-code on side A is decrypted by the ballot printer at once, so that the voter receives a ballot with a candidate list. The QR-code on side B is decrypted by the tellers during the tallying of the votes.

In this description, we omit an example and some parts of the cryptography that are very similar to the decryption mix-net version of Prêt-à-voter and focus more on the difference between these versions.

4.5.1 Setup phase

There are five different types of agents involved in the re-encryption mix-net version of Prêt-à-voter: the election authority divided into groups, the mix-servers, the tellers, the bulletin board and the voters.

- The election authority includes the set of voters, the set of candidates, the set of mix-servers and the set of tellers in the voting system. The election authority divided, into groups, generates the ballot forms.
- The mix-servers shuffle and re-encrypt the received votes.
- The tellers generate key-pairs both for encryption of ballots and decryption of votes and publish the election result.
- The bulletin board passively provides storage of information used for verification and results.
- The voter votes for the candidate of her choice, and can verify that her vote was counted correctly.

For simplicity, we assume that the election authority divided into groups, the mix-servers and the tellers are all divided into k groups. In reality, this number can be different for each types of agents.

First, the election authority divided into the set of election authorities $\mathcal{G} = \{G_1, G_2, .., G_k\}$ determines the set of voters, $\mathcal{V} = \{V_1, V_2, ..., V_l\}$, the set of candidates, $\mathcal{T} = \{T_1, T_2, ..., T_m\}$, the set of mix-servers $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ and the set of tellers $\mathcal{N} = \{N_1, N_2, ..., N_k\}$. The tellers and the ballot printer generate two independent keys using exponential ElGamal presented in chapter 2.1.2.

Key Generation using exponential ElGamal

We assume the ElGamal parameters (g, p, q) are made public in advance. p and q are large primes such that $q \mid p - 1$, and g is a generator of $\mathbb{Z}_q$ which is isomorphic to a subgroup of $\mathbb{Z}_p^*$ with order q.

- The ballot printer randomly selects a secret key $sk_r \in \mathbb{Z}_q$ and reveals its public key $h_r = g^{sk_r}$.
- The set of tellers, $\mathcal{N}$ generates the secret key $sk_t \in \mathbb{Z}_q$ in a threshold fashion. Then publishes the corresponding public key $h_t = g^{sk_t}$.

Encryption The election authority divided into groups, $G_1, G_2, .., G_k$, is in charge of the encryption using the public keys. They have to encrypt the order of the candidate list in two ways. The first encryption, using h_r is decrypted later by the ballot printer during the voting phase. The second encryption using h_t is decrypted later by the set of tellers during the tallying phase.

1. G_1 randomly selects $\alpha_1 \in \mathbb{Z}_m$ and $x_1, y_1 \in \mathbb{Z}_q$ to generate an encryption pair

$$(g^{x_1}, h_r^{x_1} \cdot g^{-\alpha_1}) \text{ and } (g^{y_1}, h_t^{y_1} \cdot g^{-\alpha_1}).$$

2. For $i \in [2, k]$:

   - G_i randomly selects $\alpha'_i \in \mathbb{Z}_m$ and x'_i, $y'_i \in \mathbb{Z}_q$ to generate an intermediate encryption pair

$$(g^{x'_i}, {h_r}^{x'_i} \cdot g^{-\alpha'_i}) \text{ and } (g^{y'_i}, h_t^{y'_i} \cdot g^{-\alpha'_i}).$$

   - G_i multiplies the intermediate onion pair with the encryption pair received from G_{i-1}

$$(g^{x_i}, {h_r}^{x_i} \cdot g^{-\alpha_i}) = (g^{x'_i}, {h_r}^{x'_i} \cdot g^{-\alpha'_i}) \cdot (g^{x_{i-1}}, {h_r}^{x_{i-1}} \cdot g^{-\alpha_{i-1}})$$

$$(g^{y_i}, {h_t}^{y_i} \cdot g^{-\alpha_i}) = (g^{y'_i}, {h_t}^{y'_i} \cdot g^{-\alpha'_i}) \cdot (g^{y_{i-1}}, {h_t}^{y_{i-1}} \cdot g^{-\alpha_{i-1}}).$$

3. The final encryption pair is: $(g^x, {h_r}^x \cdot g^{-\alpha})$ and $(g^y, {h_t}^y \cdot g^{-\alpha})$, where

$$x = x_k = x_1 + \Sigma_{j=2}^{k} x'_j \mod q$$

$$y = y_k = y_1 + \Sigma_{j=2}^{k} y'_j \mod q$$

$$\alpha = \alpha_k = \alpha_1 + \Sigma_{j=2}^{k} \alpha'_j \mod q.$$

The order of the candidate list written on the ballot is denoted by α. Note that α must be equal in both encryption.

Generating a ballot We now describe the process of generating a ballot. A ballot must include two sides, A and B. Side A contains a QR-code containing encrypted information of the list of the candidates in an alphabetical order $(T_1, T_2, ..., T_m)$ with a cyclic shift α. Side B contains the ballot number, boxes aligned with the candidate list and the QR-code containing the same information encrypted in a different way.

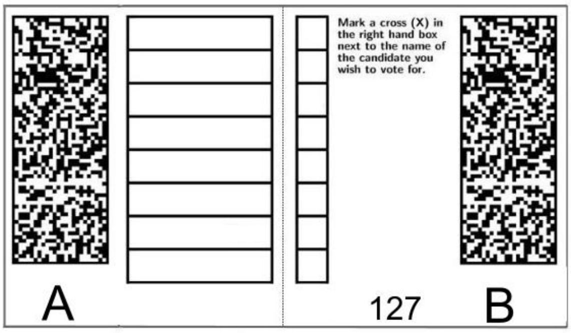
Generating a ballot

A ballot consists of side A, s^A, and side B, s^B. To create a ballot the election authority does the following:

1. Selects a unique ballot number, denoted *BN*.
2. Generates an order α of the candidate list obtaining $(T_1, T_2, ..., T_m)_\alpha$.
3. Encrypts the order of the candidate list α using h_r which is encoded into the QR$_A$-code on side A.
4. Encrypts the order of the candidate list α using h_t which is encoded into the QR$_B$-code on side B.
5. Generates the ballot, $s = (s^A, s^B)$ where, $s^A = \{$QR$_A\}$ and $s^B = \{$*BN*,QR$_B\}$.

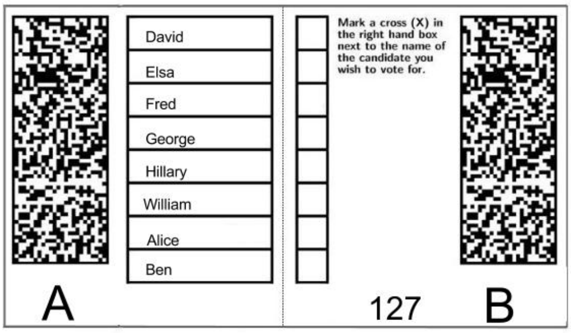
4.5.2 Voting phase

During the voting phase, the voter receives a ballot $s = (s^A, s^B)$.


Mark a cross (X) in the right hand box next to the name of the candidate you wish to vote for.
A
127
B

A ballot before the ballot printer has printed on the candidate list.

She inserts the ballot into the printer. The ballot printer decrypts the code on side A, QR$_A$ by using the secret key sk_r, and prints out the candidate list in the order $(T_1, T_2, ..., T_m)_\alpha$ corresponding to the decrypted code.


David
Elsa
Fred
George
Hillary
William
Alice
Ben
A
Mark a cross (X) in the right hand box next to the name of the candidate you wish to vote for.
127
B

A ballot after the ballot printer has printed on the candidate list.

The voter chooses the candidate of her choice, $T_i \in \mathcal{T}$, and marks a cross in the corresponding box on s^B. The vote cast is $V_{\text{cast}} = (s^B, \beta)$ where β is the position of the cross. She destroys s^A and receives the receipt, $V_{\text{receipt}} = (s^B, \beta)$.

4.5.3 Tallying phase

After the voting phase, side B of all the ballots are published on the bulletin board.

Including the mark on the ballot in the encryption The election authority must include the marks on each ballot in the encrypted code $(g^y, {h_t}^y \cdot g^{-\alpha})$ on side B so that it can be tallied. The position of the cross, denoted by β, is included in the encryption in the following way:

$$(g^y, {h_t}^y \cdot g^{-\alpha} \cdot g^{\beta}) = (g^y, {h_t}^y \cdot g^{-\alpha+\beta})$$

Mix-net In order to provide privacy, the encrypted votes $(g^y, {h_t}^y \cdot g^{-\alpha+\beta})$ are re-encrypted and shuffled in a mix-net by the mix-servers using permutation functions. This procedure is very similar to the decryption and shuffle phase in the decryption mix-net in chapter 4.2.3. However, instead of decrypting, the mix-servers re-encrypt the votes. These re-encryption do not change $-\alpha + \beta$.

Tallying Finally, the random sequence of re-encrypted votes are decrypted by the tellers using their secret key sk_t. After decryption, the tellers finally perform $-\alpha + \beta = \gamma$. The tellers obtain that the voter voted for candidate T_γ in the alphabetical ordered candidate list $(T_1, T_2, ..., T_m)$. The decrypted votes are then included in the tallying.

Chapter 5

Voter Verifiability in Prêt-à-Voter

In this chapter, we analyse further how voter verifiability can influence the privacy, verifiability and availability of Prêt-à-Voter. We analysed voter verifiability during two phases, when printing of ballots during or before the voting phase and when the voter verifies her vote on the bulletin board during the verifying phase.

5.1 Printing of ballots

The two versions of Prêt-à-Voter presented in chapter 4 print the ballots in two different ways. On one hand, with the decryption mix-net version, the ballots are already printed before the voting phase. On the other hand, with the re-encryption mix-net version, the ballots are printed on demand as a part of the voting phase. Either way, we argue that the election authority has the possibility to break the privacy and the verifiability of the system during this phase.

5.1.1 Verifiability

At the polling station, it is desirable to give the voter the possibility to check for herself that the system works correctly. In other words, she should be able to verify that the encryption on the ballot corresponds to the order of the candidate list printed. For example, in the decryption mix-net version the ballot generated is $s = (s^A, s^B)$, where $s^A = (T_1, T_2, ..., T_m)_{\alpha_{0,2}+h \bmod m}$ and $s^B = \{BN\text{,QR}\}$. She can verify that decrypting the information encoded in the QR-code gives $(T_1, T_2, ..., T_m)_{\alpha_{0,2}+h \bmod m}$.

If this verification is not possible, the malicious election authority can change a ballot by manipulating the ballot printer to print out a different order, $\alpha_{\text{malicious}} \bmod m$, than $\alpha_{0,2} + h \bmod m$ which is the order meant for the ballot. Although during this attack the malicious election authority can not control the voter so that she votes for a particular candidate, he can manipulate the ballot so that the candidate selected does not get her vote. For example, in this attack, if she chooses candidate T_i in $(T_1, T_2, ..., T_m)$ she is in reality voting for candidate $T_{i+|\alpha_{0,2}+h-\alpha_{\text{malicious}}| \bmod m}$.

For this reason, if the voter can verify the ballot at the polling station, the voting system becomes more secure. She can do this by asking the system to print out a test ballot. This procedure can be done as many times as she wants. However, a malicious election authority can analyse the way voters verify the system and use this to manipulate the printer. It does seem that if it is a voluntary verification procedure, the malicious election authority can get away more easily. We therefore propose two solutions permitting to increase verifiability during this phase.

A cut-and-chose protocol can be implemented for verification during the printing phase of the ballots. Cut-and-chose is a protocol between two agents. In the printing of ballots, the election authority tries to convince a voter or a neutral third party that the ballots printed are honestly constructed. This can be done in the following way: the election authority does the cut by proposing which ballots to verify and the voter chooses exactly which one she wants to verify. A solution based on the cut-and-chose protocol and similar to Demos is that all $V_j \in \mathcal{V}$ receives two ballots $s_0 = (s_0^A, s_0^B)$ and $s_1 = (s_1^A, s_1^B)$ during the voting phase. The voter chooses randomly $i \in \{0, 1\}$. Ballot s_i can be used to verify the whole procedure at the polling station. Ballot s_{1-i} is used as before, to vote and to verify the vote. Note that this solution goes against the availability of the system as it demands more of the voter, which should be as minimal as possible. Furthermore, it increases the cost of the election.

Therefore, another possible solution is to randomize the verifying process. In addition to the verification asked by interested voters, the system randomly asks a number of voters to verify a test ballot. Because the ballots chosen to be verified are picked at random, the election authority can not make analysis about the verifying procedure to attack the system. This solution also affects the availability of the system. Indeed, this is a time-consuming and demanding procedure for voters chosen to verify. It is therefore important that the amount of voters selected to verify a test ballot is well-chosen.

For both solutions, a verifying machine must be used and this machine must have access to the secret keys of mix-servers (for the decryption mix-net version) or the secret keys of tellers (for the re-encryption mix-net version) in order to decrypt the information inside the QR-code. This verifying machine must be used only for verifying purposes and should be independent to the ballot printer.

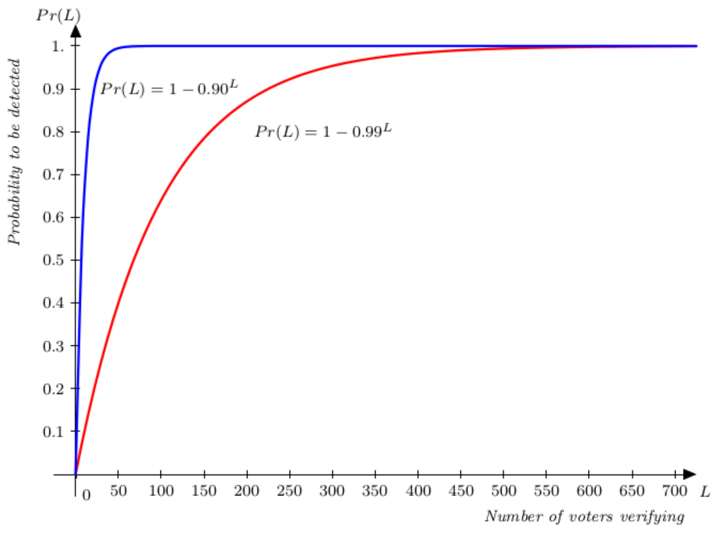
We finally make an analysis of the number of verified ballots needed to make the system somewhat trustworthy. We recall the set of voters, $\mathcal{V} = \{V_1, V_2, ..., V_l\}$ and assume that $L \in [1, l]$ is the number of voters picked randomly to help verify the system. Further, we look at two cases where the malicious election authority has randomly manipulated 1% and 10% of the ballots. The probabilities of detecting these attacks are,

$$Pr[\text{detecting the attack}] = 1 - (0.99)^N$$
$$Pr[\text{detecting the attack}] = 1 - (0.90)^N,$$

respectively. Hence, if 100 ballots are verified, the detection probability is,

$$Pr[\text{detecting the attack}] = 1 - (0.99)^{100} \approx 0.63.$$

If the percent of manipulated ballots is higher, the probability of detecting the attack obviously increases.



We observe from the graph that if the attack is done on randomly chosen ballots, we detect the attack with probability 0.99 by choosing $L = 500$. We conclude that verifying the ballots at the polling station make the system more trustworthy, only slightly affecting the availability requirement.

5.1.2 Privacy

In the traditional Norwegian voting system, for each party the ballots contains exactly the same information. It is not possible to differentiate between two ballots. By introducing voter verifiability into the voting system, Prêt-à-Voter produces distinct ballots with different order of the candidate list and a unique ballot number. That is, there exists a unique $\{(\alpha_{0,2} + h), BN\}$ for each $V \in \mathcal{V}$. The uniqueness of each ballot can therefore be used to break privacy during the printing phase.

Indeed, in the decryption mix-net version, the ballots are printed in advance. Even if the list of candidates is in different order for each ballot to prevent coercing and vote selling, the election authority have access to these ballots during the lapse of time between printing and voting. The malicious election authority can for example break the privacy by memorizing lists from ballots. Additionally, the ballots can be organized so that for example the ballots with the same order of the candidate list are in groups. Either way, these attacks destroy the anonymity provided by the cyclic shift computation during the generation of ballots and therefore can break the privacy of many voters.

The attack described above is not relevant when the ballots are printed on demand. In other words, this does not affect the re-encryption mix-net version.

Conclusion Because of the uniqueness of each ballot, the printing phase of ballots is more complicated in Prêt-à-Voter than with the traditional voting system. This opens up for several attacks both against privacy and verifiability. The printing procedure must therefore be well organised so that the malicious election authority does not obtain information of how voters verify ballots and with which ballots the voters vote with.

5.2 Verifying phase

After voting, the voter can check that her vote has been counted correctly by verifying with her receipt on the bulletin board. This provides security on the voting system. We formulate some problems that can occur and break the verifiability, privacy and availability of Prêt-à-Voter during this phase.

5.2.1 Verifiability

First of all, the verifying phase only improve the security of the system if it is done by many voters. Indeed, if only a few verify their votes, malicious election authority, mix-servers and tellers can manipulate ballots without being detected. What percent of voters that verify their vote is needed before the voting system is trustworthy? Voters behavior should be analysed to see how it affects the system. If voters are not interested in verifying, because of the weaknesses presented above, Prêt-à-Voter is probably not the best system to implement in an election.

Further, giving the opportunity to the voter to verify her vote comes with complications. Not only honest voters can challenge the system but dishonest voters also have the power to disturb it. The system must be prepared and take into consideration voters and losing candidates trying to sabotage the election by falsely stating their vote has been tallied incorrectly.

For example, a dishonest voter can change the appearance of his receipt by moving the place of his mark such that instead of (s^B, β), where β is the position of the cross, the dishonest voter replace it with $(s^B, \beta_{\text{fake}})$.

Another example is that a voter can challenge the system by saying that her vote is not included in the system. For instance, if a voter has access to a ballot not used, she can mark a cross and destroy side A and use $(s^B_{\text{fake}}, \beta)$ as a fake receipt. She can show the election authority that she does not have access to the website with her receipt.

These attacks are not problematic if done by one voter. The problem is for example when many voters of a losing party do these attacks together. In the worst case, the election may risk to abort and in the best case Prêt-à-Voter as a voting system looses credibility. Therefore, it is crucial that the system differentiates between honest and dishonest voters during the verifying phase. It is quite challenging since the election authority must inspect where the mistakes occurred without breaking the privacy of the voter. The system must provide privacy also during this critical part of the process. More verifiability should not give less privacy.

In both situations, a solution can be that during the voting phase, the voter receives a printed duplicated version of her receipt. This way, the election authority avoids receipts that can be changed manually by voters. Prêt-à-Voter should anyway mark the votes included in the system during the voting phase with an official stamp so that unused ballot can not be used as a fake receipt.

5.2.2 Privacy

The voter verifiability is based on the fact that the voter actually has a receipt, (s^B, β). This receipt does not give away any information about how the voter has voted but give at least the information that the voter has voted. Consequently, a voter can be forced to vote by a coercer. It does not seem like a problem in a national election in Norway, but it becomes a bigger problem when the set of candidates is small. By introducing voter verifiability, Prêt-à-Voter uncover more information about the vote than the traditional Norwegian voting system where voters leave without any proof of their voting.

5.2.3 Availability

Many difficult cryptographic elements are not verified by the voter who only partially verifies the system. For example, the mix-net decryption or re-encryption by randomized partial checking that we analyse in chapter 6 is verified by neutral parties not by the voters. The only thing a voter can verify is that the receipt is on the bulletin board, the rest must be trusted. In the traditional voting system, the voter verifies that the vote is put in the urn. For the voters, the verifying phase can be seen as the same process as with the traditional voting system only in a different form. So one can argue that from the voters perspective, the voter might not be convinced that her vote has been counted correctly. In other words Prêt-à-Voter has not yet succeeded in finding a proper and available way for the voter to verify her vote.

Chapter 6

Testing cryptographic components in Prêt-à-Voter

All the components used by Prêt-à-Voter are secure if used honestly and correctly by the different agents. Therefore, for each step of the system, the election authority and the mix-servers must submit proofs and evidences of their honest behaviour. These must not be overlooked and represent a significant part of a well-designed and trustworthy system.

On the other hand, the system can not afford aborting the election for every error as it goes against the availability property. The system must therefore test thoroughly components that are reported to make mistakes. The causes of the errors must be found and, if needed, the system must be changed. For some components, the system can hypothetically allow some mistakes. For these, the challenge is therefore to estimate an error bound, in other words to find how many errors can occur without affecting the outcome of the election. The system must either way clearly present a procedure to follow in case of errors occurring, and these must be decided before the beginning of the election.

In this chapter, we analyse the following cryptographic components of Prêt-à-Voter: Randomized Partial Checking, length of ciphertext during mix-net and proof of shuffling. These are analysed first separately and then together in order to give an overall conclusion on the verifiability, privacy and availability properties of Prêt-à-Voter. This chapter concentrates its analysis on only some components. But, in order to give a proper analysis of the system, each and every component of the system should be analysed.

Bulletin Board Prêt-à-Voter makes some assumptions around the bulletin board. Indeed, the system assumes that the bulletin board posts publicly information without having the possibility to delete or change information already posted. This is a strong assumption and if it does not hold, there are several ways to break privacy, verifiability and availability. These weaknesses should therefore be formally analysed and a proper bulletin board should be created.

6.1 Randomized Partial Checking

We repeat that during the tallying phase, each mix-server S_i receives a list $L_{i,2}$. The mix-server then decrypts or re-encrypts all the votes $B^j_{i,2} \in L_{i,2}$ for $j \in [1, l]$ twice and shuffles between each time to obtain $L_{i-1,2}$.

Prêt-à-Voter uses Randomized Partial Checking, RPC, to provide strong evidence that the decryption or the re-encryption of the votes during this phase went correctly. In other words, it ensures that no votes have been deleted, added or replaced during the mixing. It is important to note that Prêt-à-Voter uses RPC as a proof checked by neutral third parties and is not a part of the voter verifiability.

RPC challenges each mix-server $S_i \in \mathcal{S}$ where $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ is the set of mix-servers, to reveal the source and destination of a subset of the list $L_{i,1}$ during the mixing. This way, one can be certain that this subset has not been changed. Repeating the procedure for each mix-server assures higher level of security as the probability of changing ballot gets lower.

The number of votes $B_j \in L_i$ verified each time should be:

- Not too high such that the system still ensures privacy. Indeed, if all the mix-servers reveal the source and destination of the same vote, then this vote is not anonymous anymore. (Privacy)
- Not too low since if RPC only verifies a few votes each time, malicious mix-servers have the possibility to get away with manipulating votes. (Verifiability)

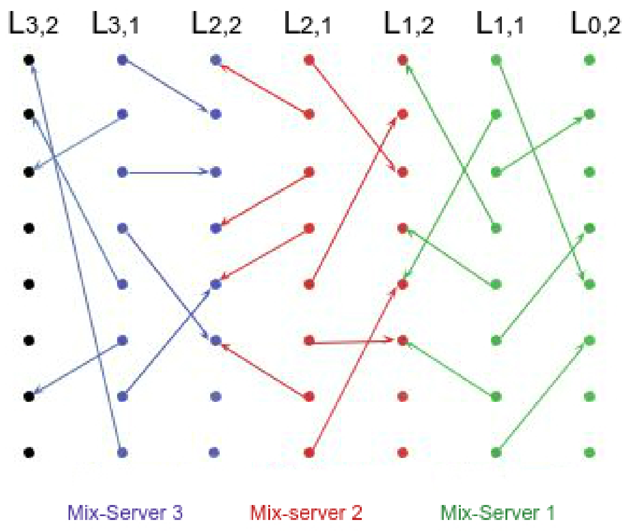
The mix-server S_i publishes on the bulletin board its decrypted or re-encrypted version in between each shuffle, that is $L_{i,1}$ and $L_{i-1,2}$. At the end of the mix-net, S_i is asked to reveal the link between $L_{i,2}$ and $L_{i,1}$ of half of the votes B^j_i and the link between $L_{i,2}$ and $L_{i-1,2}$ for the other half of the votes. Then, for the next mix-server S_{i-1}, already half of its received votes $B^j_i \in L_{i-1,2}$ have their sources revealed. Therefore, he is asked to show the destination of half of these and half of the rest.

Randomized Partial Checking procedure This procedure takes place after the k mix-servers have published their computations. RPC analyses the first output $L_{i,1}$ of each mix-server S_i in the following way.
Mix-server k:

1. Is asked to reveal the source of half of his output $L_{1,1}$. We name this subset S_1.
2. Is asked to reveal the destination of the rest of $L_{1,1}$. We name this subset D_1.

Mix-Server j for all $j \in [1, k-1]$:

1. Is asked to reveal from $L_{j,1}$ the source of half of D_{j-1} and half of S_{j-1}. These together are named S_j.
2. Is asked to reveal the destination of the rest of $L_{j,1}$. We name this subset D_j.



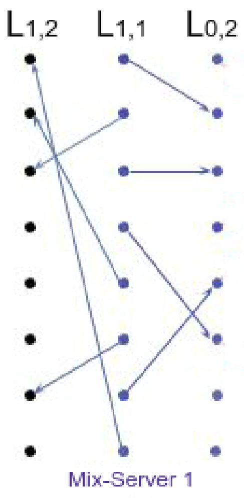
Randomized Partial Checking: Illustration with 3 mix-servers and 8 votes

RPC has the quality of being a very fast and efficient procedure, which is important since the results of an election must appear as soon as possible. In other words, the availability requirement is satisfied. However, this system has some weaknesses both in privacy and verifiability.

6.1.1 Privacy

First of all, a large enough number of mix-servers is needed to ensure privacy in a mix-net if using RPC. Indeed, in order to verify the system, RPC actually reduces some of the privacy given by the mix-net shuffling. If the mix-servers are few, the link revealed between each list L can be used to trace votes. In this case, it is important to note that one can obtain only partial information about a certain vote, and not be able to know its whole journey. This can be illustrated with one mix-server which shuffles twice.

We illustrate below one mix-server that reveals partial information about a vote. Each vote, can be narrowed down to be part of 50% of the final result instead of 100%.



Randomized Partial Checking: Illustration with 1 mix-server and 8 votes

Pfitzmann attack We describe Pfitzmann attack [10] on the privacy requirement when using RPC on re-encryption mix-net. In this attack, the mix-server S_k which is the first one to re-encrypt, is the adversary. He can attack one voter by knowing how she votes. We explain this attack with the following example.

1. The mix-server S_k receives the list $L_{k,2} = (B^1_{k,2}, B^2_{k,2}, ..., B^l_{k,2})$.

2. He chooses to attack the vote $B^3_{k,2}$.

3. He chooses a random δ and computes $(B^3_{k,2})^\delta$.

4. He throws away $(B^7_{k,2})$ and replaces it by $(B^3_{k,2})^\delta$. *He can get away with this replacement with a probability of 0.5.*

5. He re-encrypts and shuffles the list $L_{k,1} = (B^1_{k,1}, B^2_{k,1}, B^3_{k,1}, ..., (B^3_{k,1})^\delta, ..., B^l_{k,1})$ before sending it on to the mix-server S_{k-1}.

6. When the decryption is done, the plaintexts $(m^1, m^2, ..., m^l)$ are posted.

7. S_k investigates which $m^k = (m^j)^\delta$.

8. S_k knows that m^j is the plaintext corresponding to $B^3_{k,2}$.

This attack is possible because of the homomorphic properties in the re-encryption mix-net. Indeed, if B and B^δ are two ciphertexts. Then, $D(B)$ and $D(B^\delta)$ are the plaintexts respectively. Because of the homomorphic property we have $D(B^\delta) = D(B)^\delta$.

Now, we analyse the probability of detecting this attack during RPC. There is a 50% chance to detect that one ciphertext B^j_i has been manipulated. If the malicious mix-server

S_k manipulates several ciphertexts in a similar way, his risk of getting caught increases. If he breaks the privacy of t voters, the probability of detecting this attack is,

$$Pr[\text{detecting the attack}] = 1 - (0.5)^t.$$

The malicious S_k can reduce this probability by improving his attack. In our example, the original cipher $B^7_{k,2}$ is now replaced by the product of t other ciphertexts each taken to different powers δ_j. That is, $B^7_{k,2}$ is replaced by for example,

$$B^7_{k,2}{}' = (B^1_{k,2})^{\delta_1} \cdot (B^2_{k,2})^{\delta_2} \cdot ... \cdot (B^t_{k,2})^{\delta_t}.$$

Because of the homomorphic property we have that,

$$D(m_{k_1}^{\delta_{k_1}} \cdot m_{k_2}^{\delta_{k_2}} \cdot ... \cdot m_{k_t}^{\delta_{k_t}}) = D(m_{k_1})^{\delta_{k_1}} \cdot D(m_{k_2})^{\delta_{k_2}} \cdot ... \cdot D(m_{k_t})^{\delta_{k_t}}.$$

Hence, the malicious mix-server S_k can find the combination satisfying this property and break the privacy of t voters. Because he only changes one vote $B^7_{k,2}$, the chance of being caught is still 50%.

Finding a solution to Pfitzmann attack for re-encryption mix-net is quite difficult. One idea can be to treat the mix-server S_k differently from the others. The system can, for example, ask him to show all his computation without shuffling. The malicious mix-server S_k can not produce Pfitzmann attack anymore. But, this solution only moves the problem to the next mix-server S_{k-1} as he now has the power to produce this attack. Hence a better solution can be to divide the task of S_k by splitting his work in a threshold fashion, but this raises new difficulties. We conclude that this attack is too powerful on RPC, consequently RPC should not be used with re-encryption mix-net.

6.1.2 Verifiability

RPC shows strong evidence that the ballots are cast and counted correctly, but is in no way a proof. This might not be a problem in an election where the results are very different for each party, but becomes a problem when the result differs with only a few votes. How can RPC ensure that these few votes have not been changed during the process? Cheating with a small amount of votes during the process can suddenly change the result of the election.

Duplicate a vote attack This is an attack on the verifiability requirement when using RPC on re-encryption and decryption mix-net. The malicious mix-server S_i duplicates and throws away ciphertexts. The number of duplicated ciphertexts must be equal to the number of ciphertexts thrown away.

1. The mix-server S_i receives the list $L_{i,2} = (B^1_{i,2}, B^2_{i,2}, ..., B^l_{i,2})$.
2. He chooses to attack the vote $B^3_{i,2}$, and to duplicate the vote $B^7_{i,2}$.
3. The re-encryption or decryption of $B^3_{i,2}$ is replaced by the re-encryption or decryption of $B^7_{i,2}$.

4. The mix-server publishes $L_{i-1,2} = (B^1_{i-1,2}, B^2_{i-1,2}, B^7_{i-1,2}, ..., B^7_{i-1,2}, ..., B^l_{i-1,2})$ permuted, instead of $L_{i-1,2} = (B^1_{i-1,2}, B^2_{i-1,2}, B^3_{i-1,2}, ..., B^7_{i-1,2}, ..., B^l_{i-1,2})$.

Once again, during RPC, there is a 50% chance of a B^j being checked. The malicious mix-server S_i can be detected only if RPC asks for the source of both $B^7_{i-1,2}$. If RPC asks for the re-encryption or decryption proof of one but not both, his cheating remains undetected. The probability is therefore,

$$Pr[\text{sources of both } B^7_{i,1} \text{ are checked}] = 0.5 \cdot 0.5 = 0.25.$$

That is, the malicious mix-server is not detected 75% of the time. To generalise the situation, a mix-server can duplicate-and-throw-away t pairs of all the votes with the probability of being detected,

$$Pr[\text{detecting the attack}] = 1 - (0.75)^t.$$

A simple solution to the duplicate attack is to forbid mix-servers to produce several equal ciphertexts. But, with decryption mix-net, this solution does not work for the mix-server S_1 since he decrypts to several equal plaintexts. So, when asked to prove some of the plaintext, he has to find one appropriate ciphertext that satisfy this result. He does not have to prove a one-to-one correspondence like the mix-servers $S_i \in [2, k]$. Therefore, a malicious mix-server S_1, can organise this attack by still having enough distinct proofs for each vote. Consequently, it seems reasonable to treat mix-server S_1 differently. We present some solutions to prevent mix-server S_1 to cheat.

- The work of mix-server S_1 can be shared in a threshold fashion. However, this is a complicated and expensive solution. Furthermore, we want to avoid situations where the mix-servers must work together.

- Each vote could have a distinct mark so that the plaintexts have a uniqueness property. This solves the problem as mix-server S_1 can not cheat with the proofs. However, this cause some privacy issues since now each ballot is marked uniquely, allowing coercing.

- Mix-server S_1 is requested to decrypt without shuffling and to give a complete proof of decryption during his last computation. This appears to a good solution but it removes one shuffling. This has to be taken into consideration so that there are still enough shuffling to provide privacy.

Conclusion We conclude that RPC is a solution that needs improvement in order to be used during an election. Some behaviour must be forbidden and the first and last mix-server should be treated differently to avoid attacks such as the Pfitzmann attack and duplicate attack. Instead of RPC, Prêt-à-Voter can use a proper shuffle with a proper verification proof. These proofs might be less available, but at the expense of availability, a proper shuffle can fulfill the privacy and verifiability requirements.

6.2 Length of Ciphertext

As described in chapter 6.1, mix-net permits the shuffling of the ballots providing anonymity to each vote. We now demonstrate that this component can actually fail to satisfy privacy if the length of ciphertext differ from vote to vote.

6.2.1 Privacy

We note that secure encryption schemes used in Prêt-à-Voter such as exponential El-Gamal do not hide the length of the ciphertexts. During mix-net, each mix-server must publish his output on the bulletin board. If the length of a ciphertext is different from all the others, the journey of the vote can be traced.

During decryption mix-net, the last output is the plaintext. Therefore, almost the whole journey of a vote with special length can be retrieved. The privacy of this vote is not entirely broken but weakened. The first mix-server S_1 is the one doing the last computation and consequently has access to the whole journey of this vote with special length. Only he has access to the last part of the journey. He can therefore be a potential adversary in this situation.

Furthermore, if RPC is used to verify the mix-net, the problem becomes even bigger. Indeed, the neutral third party during RPC asks 50% of the destination of the votes. A malicious neutral third party can therefore ask about the destination of the vote with the special length to obtain its whole journey, and hence break the privacy of this vote. Normally, the neutral third party must choose randomly which input and output to ask for, but then he still has 50% chance of obtaining this information. The adversaries are then both the first mix-server S_1 and neutral third party.

Finally, since the first mix server S_1 should be treated differently during RPC, we concluded earlier in chapter 6.1 that he should not do the shuffling during the last part but just give a proof of all his outputs. We see now that this solution does not work anymore. Indeed, if the first mix-server does not shuffle the vote the last time, everybody has access to the whole journey of the special vote, which breaks entirely the privacy of this vote.

Hence, to preserve privacy, it must be impossible to trace a vote by the length of its ciphertext. This can be done using a padding method.

Padding refers to a number of different practices where one changes the length of a plaintext in order to encrypt. If the size of an encrypted message gives some information away, one can add a random number of padding bytes to the message. In other words, the agent that encrypts the messages adds a random number of padding bytes to each message, making the length of the plaintext random. The last byte indicates the number of random bytes that is added, so that during decryption, one can find the initial plaintext.

We investigate how the ciphertexts in Prêt-à-Voter can have different lengths. The initial value $\alpha_{0,2}$ is randomly chosen from $\mathbb{Z}$ for each ballot. These initial values, different for each ballot are encrypted with the same public keys $(n_{i,1}, b_{i,1})$ and $(n_{i,2}, b_{i,2})$ generated by each mix-server $S_i \in \mathcal{S}$. Since each initial plaintext pair, $(\alpha_{0,2}, t_{1,1})$, is encrypted with the same public keys, it is the length of $\alpha_{0,2}$ that can affect the length of the ciphertext. The attack presented above can therefore occur.

We introduce a solution to prevent this attack. By using the padding method, the election authority can add a random number of padding bytes to the initial plaintext, $(\alpha_{0,2}, t_{1,1})$, and continue to do this between each layer of encryption. Then the lengths of ciphertexts is completely random and malicious agents can not trace one encrypted vote during its decryption.

6.3 Shuffle proof

6.3.1 Privacy

Prêt-à-Voter uses shuffling during at least two different phases of the system. First, to permute the candidate list for each ballot. Prêt-à-Voter based the order of the candidate list in fact on a cycling shift. But, this solution does not work if the voter can vote for multiple candidates. In a cycling shift, the list of candidates is always in the same order but starting at a different candidate. This means the relative positions of candidates remains constant. This can give away the way the voter votes. That is why, in order to vote for several candidates instead of just one, the system must use permutation shift instead of a cyclic shift. This permutation must be completely random so that it does not break the privacy of the voter and the set of candidates must obviously remain the same after permutation.

Second, to permute the votes during decryption or re-encryption mix-net as seen earlier. The privacy property of the mix-net shuffle is based on a permutation in between each computation. This permutation must occur in a completely random way. If not, mix-servers can work together to obtain information about some votes.

These two cases are quite different. The first one, does a simple permutation on a same set but the second one does a permutation while also changing the appearance of the set. The later one is therefore more complicated to prove. The system must therefore provide proof or strong evidence that this permutation occurs correctly. We are left with the following questions.

- How to prove randomness during permutation.
- How to prove that the permutation is actually a proper permutation in the case where the input and output are the same representation of the same data.
- How to prove that the permutation is actually a proper permutation in the case where the input and output are different representations of the same underlying data?.

The cycling shift (or the permutation shift if multiple candidates voting) is central in Prêt-à-Voter. The system is based on the randomness of these shifts and it should not be possible to link the ballot number BN to a specific order α. Unfortunately, to verify this randomness is extremely difficult. Indeed, how does one prove that a permutation does not respect any rules. Instead of a proof, we can test a subset of ballot $s_i = (s_i^A, s_i^B)$ where $i \in [1, l]$

and use statistical analysis to look at the distribution of the different cyclic shift and the relation with the corresponding ballot number BN.

A solution for the mix-servers shuffling is to provide a proper shuffle instead of RPC as seen in chapter 6.1. To prove this, we can first look at some properties that the permutation holds. We assume that one list of elements is permuted such that,

$$(B^1, B^2, ..., B^l) \to (B^{\pi(1)}, B^{\pi(2)}, ..., B^{\pi(l)}).$$

Proof of permutation can be based on the fact that the sum and product of the two lists remain the same,

$$\Sigma_{i=1}^{l}(B^i) = \Sigma_{\pi(i)=1}^{l}(B^{\pi(i)}) \quad \text{and} \quad \Pi_{i=1}^{l}(B^i) = \Pi_{\pi(i)=1}^{l}(B^{\pi(i)}).$$

We present the iterated logarithm multiplication problem [12]. This is a simpler version then proving shuffling with re-encryption or decryption but it is meant to give us a general understanding on how to proceed when proving shuffling.

Let $(X_1, X_2, ..., X_n)$ and $(Y_1, Y_2, ..., Y_n)$ be two sequences that are publicly known. Furthermore, $X_i = g^{a_i}$ and $Y_i = g^{b_i}$, $\forall\ 1 \leq i \leq n$. We want to prove that

$$g^{a_1 a_2 ... a_n} = g^{b_1 b_2 ... b_n},$$

without giving information about a^i and b^i. We present a protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, proving the iterated logarithm multiplication problem.

Iterated Logarithm Multiplication Proof Protocol

1. (a) $\mathcal{P}$ generates $c_1, c_2, ..., c_n \in \mathbb{Z}_q$ randomly and secretly.

   (b) $\mathcal{P}$ computes

$$\begin{aligned} A_1 &= Y_1^{c_1} = g^{b_1 c_1} \\ A_2 &= X_2^{c_1} Y_2^{c_2} = g^{a_2 c_1 + b_2 c_2} \\ &... \\ A_i &= X_i^{c_{i-1}} Y_i^{c_i} = g^{a_i c_{i-1} + b_i c_i} \\ &... \\ A_n &= X_n^{c_{n-1}} = g^{a_n c_{n-1}} \end{aligned}$$

   (c) $\mathcal{P}$ sends the sequence $(A_1, A_2, ..., A_n)$ to $\mathcal{V}$.

2. (a) $\mathcal{V}$ generates randomly $d \in \mathbb{Z}_q$ as a challenge.

   (b) $\mathcal{V}$ sends the challenge d to $\mathcal{P}$.

3. (a) $\mathcal{P}$ must compute $r_1, r_2, ..., r_{n-1} \in \mathbb{Z}_q$ such that

$$\begin{aligned} X_1^d Y_1^{r_1} &= A_1 \\ X_2^{r_1} Y_2^{r_2} &= A_2 \\ &\dots \\ X_i^{r_{i-1}} Y_i^{r_i} &= A_i \\ &\dots \\ X_n^{r_{n-1}} Y_n^d &= A_n \end{aligned}$$

   (b) $\mathcal{P}$ sends the sequence $(r_1, r_2, ..., r_{n-1})$ to $\mathcal{V}$.

4. $\mathcal{V}$ verify that the sequence $(r_1, r_2, ..., r_n)$ satisfies the equations and accepts the proof if it holds.

We explain how the move (3) by the prover is effectuated.

$X_1^d Y_1^{r_1} = A_1$ can be reduced to the equation $b_1 \cdot c_1 = a_1 d + b_1 r_1$.
$X_2^{r_1} Y_2^{r_2} = A_2$ can be reduced to the equation $a_2 \cdot r_1 + b_2 \cdot r_2 = a_2 \cdot c_1 + b_2 \cdot c_2$.
...
$X_i^{r_{i-1}} Y_i^{r_i} = A_i$ can be reduced to the equation $a_i \cdot r_{i-1} + b_i \cdot r_i = a_i \cdot c_{i-1} + b_i \cdot c_i$.
...
$X_n^{r_{n-1}} Y_n^d = A_n$ can be reduced to the equation $a_n \cdot r_{n-1} + b_n \cdot d = a_n \cdot c_{n-1}$.

The honest prover therefore has to solve a system of n equations with n unknowns. If he is dishonest the probability to not being caught by the prover is $1/q$. We refer to the article of Neff [12] for a complete proof of this protocol with respect to completeness, soundness and special honest-verifier zero-knowledge.

Conclusion We conclude that Prêt-à-Voter must provide proofs on the shuffling. Ideally, these proofs must show both the randomness of the permutation and that the permutation does not change the elements. Without proofs, malicious election authority can break the privacy of voters. We propose for the shuffling during mix-net to use a proof based on Iterated Logarithm Multiplication Proof Protocol instead of RPC. Secondly, we propose to do some statistical analysis on ballots. This way, we can have a better control over the randomness of the candidate list on each ballot.

Chapter 7

Discussion around the concept of privacy

We described both Prêt-à-Voter and Demos as voting system based at the polling station, although both have variants as remote voting systems. Remote voting systems are even more subject to coercing and consequently more complicated to create. In this section, we present several definition on coercion-resistance that can be found in literature on voting systems. We then present a mathematical definition on privacy based on Demos privacy game. Finally, we show different attacks on privacy in Prêt-à-Voter and Demos.

The term coercion resistance is most relevant in the voting phase. The voter can be coerced by an adversary or can also cooperate with the adversary by selling votes. There are a lot of different ways and reasons to coerce a voter such as forcing her to vote for a specific candidate, to vote or to not vote at all.

7.1 Definition of coercion resistant

We have chosen in this thesis the following definition of coercion-resistant. “A coercer can not be certain whether the voter cooperated with him, even if they interact while she votes.” We list below some definitions found in the litterature.

- “Coercion resistance requires that no adversary can learn any more about votes than is revealed by the results of tabulations” [6].
- “A voter cannot cooperate with a coercer to prove to him that she voted in a certain way” [7].
- “We define a scheme to be coercion-resistant if it is infeasible for the adversary to determine if a coerced voter complies with the demands” [9].

- "A coercion-resistant voting system is one in which the user can deceive the adversary into thinking that she has behaved as instructed, when the voter has in fact cast a ballot according to her own intentions" [9].

All these definitions imply that the coercer coerces if he is certain of the outcome of the vote of the voter. One can argue that a definition could include the case where the coercer has strong evidences. For example, if a voter takes a picture in the polling booth of her ballot and shows it to her coercer, the coercer can not be certain that it is exactly the vote she casts. However he can be confident that she followed his demands. Moreover, under a remote voting system, the voter can be influenced more easily by members of family and surrenders to vote in a certain way. Should strongly influencing be included in the definition of coercion resistant even in a smaller level? The different levels of coercing such as strongly influenced, having strong evidence and being sure that a voter votes a certain way should be taken into consideration in the definition.

7.2 Mathematical definition

It is essential for a voting system to have a mathematical model that can support the definition of coercion resistant. This way a system can be tested on this requirement in a concrete manner. A simple mathematical definition can be motivated as followed. Let V_1 and V_2 be two voters and V_{cast1} and V_{cast2} be two cast votes. We say that a system is coercion resistant if the coercer is not able to distinguish between the case where V_1 voted with V_{cast1} and V_2 voted with V_{cast2} and the case where V_1 voted with V_{cast2} and V_2 voted with V_{cast1}. For a more advanced definition, articles about voting systems often use models based on game theory. In this chapter we present the privacy game used in Demos [11].

7.2.1 Privacy game

The privacy game is between an adversary and a challenger. The Adversary is an external agent trying to obtain some information about the voters. The challenger is the opponent in the game, in charge of different phases of the voting system. The challenger also plays the role of the voters that are not coerced. The adversary wins if he distinguishes between a real and a fake voting phase from a specific voter supplied by the challenger. A voting phase is the view of how a voter votes. Before further explanation, we present the notations of the game.

- The external agent is the adversary denoted by $\mathcal{A}$.
- The challenger is denoted by $\mathcal{C}$.
- The set of corrupted voters is denoted by $\mathcal{V}_{\text{corrupt}}$.
- The set of receipts of honest, successful votes is denoted by $\bar{\mathcal{V}}_{\text{receipt}}$.
- The maximum number of voters that are corrupted is denoted by L.

Flipping a coin b The challenger starts the game by flipping a coin $b \in \{0,1\}$. The behaviour of the challenger, depends on b. First, it decides whether the challenger votes for candidate T_0^* or T_1^*. Secondly, the outcome of b decides whether the challenger is honest with the adversary or not. If $b = 0$ the challenger gives the real voting phase. If $b = 1$ he tries to cheat the adversary and gives the fake voting phase.

Now we present the privacy game between the adversary, $\mathcal{A}$ and the challenger, $\mathcal{C}$. The challenger does the computation of the election authority.

Privacy Game

1. $\mathcal{A}$ selects the set of voters, $\mathcal{V} = \{V_1, V_2, ..., V_l\}$ and the set of candidates, $\mathcal{T} = \{T_1, T_2, ..., T_m\}$.
2. $\mathcal{C}$ flips a coin $b \in \{0,1\}$ and does the work of the election authority during the setup phase.
3. For each voter V_i, $\mathcal{A}$ chooses to corrupt the Voter or not;
    - If $V_i \in \mathcal{V}_{\text{corrupt}}$, $\mathcal{A}$ plays the role of V_i and $\mathcal{C}$ plays the role of the election authority and bulletin board.
    - If $V_i \notin \mathcal{V}_{\text{corrupt}}$, $\mathcal{A}$ provides $\mathcal{C}$ with two ways of voting, T_0^* or T_1^*, for V_i. $\mathcal{C}$ votes on behalf of V_i for the candidate T_b^*, and after casting the votes provides $\mathcal{A}$ with V_{receipt} and:
        - If $b = 0$, $\mathcal{C}$ gives the casting phase of the voter V_i.
        - If $b = 1$, $\mathcal{C}$ gives the fake casting phase of the voter V_i.
4. $\mathcal{C}$ does the tallying of all the votes and publishes the results on the bulletin board.
5. $\mathcal{A}$, with all the information obtained during the game, outputs $b' \in \{0,1\}$.

Privacy game$(m,l) = 1$ if:

- $b = b'$.
- $\mid \mathcal{V}_{\text{corrupt}} \mid \leq L$.
- The result does not leak b.

The adversary wins if Privacy game$(m,l) = 1$.

The challenger wins if Privacy game$(m,l) = 0$.

By using this privacy game, we present the mathematical definition of privacy.

Privacy Definition

The system supports privacy if, for $0 < \epsilon << 1$, $l, m, L \in \mathbb{N}$ and at most L corrupted voters:

$$| \; Pr[\text{Privacy game}(m, l) = 1] - 1/2 \; | \leq \epsilon$$

Demos fulfills the privacy requirement if the probability that the adversary distinguishes between a fake way of voting and a real way of voting is negligible. A similar privacy game can be used for Prêt-à-voter.

7.3 Prêt-à-voter and Demos

The privacy game described situations where the voter can gives a real or a fake casting phase. In this chapter, we look at different attacks both in Prêt-à-voter and Demos where the coercer obtains information of the cast vote.

Before showing some possible attacks on the privacy of paper-based electronic voting systems, we want to point out that it is possible for a voter to be coerced in the traditional Norwegian system. Indeed, at first sight it may seem that the conventional systems with on-site voting may fulfill this privacy requirement, but with use of new technologies new threats on the voter privacy can occur. The voter can be forced to use her cellphone or other small electronic devices to vote in a certain manner during the voting phase. Schafferand and Schedler describes some concrete examples on how "parties in the Philippines give out carbon paper so voters can copy their ballots, whereas Italian parties lend mobile phones with cameras so reward recipients can photograph how they vote" [16].

Italian attack The voter is forced to mark her vote in a certain way. Since the voter obtains the receipt $V_{\text{receipt}} = (s^B, \beta)$, the coercer only needs to verify β. It is important to see that the coercer does not force the voter to vote for a particular candidate but instead does not allow her to vote as she wishes.

Solution At the polling station, the voter can choose between different ballots such as the position β actually corresponds to the one she wants to vote for.

Force to vote attack The voter is obliged to vote by the coercer. Since the voter walks out with the receipt V_{receipt} the coercer can verify that her receipt is part of the system on the bulletin board.

Solution A solution is also difficult to find as the V_{receipt} is crucial during the verifying phase. We note that this attack is not relevant in a country such as Norway with many candidates and with the possibility to vote blank.

Obtaining both sides of the ballot attack The voter votes with $s = (BN, s^A, s^B)$ and walks out with both s^A and s^B. With these two sides available, the coercer has all the information needed to verify that the voter followed his demands. This attack also permits voters to sell their votes. we find this attack relevant for both Prêt-à-voter and Demos.

Chain attack The coercer asks a voter V_1 to walk out from the polling station with an unused ballot $s_1 = (BN_1, s_1^A, s_1^B)$. She can give it to a coercer and let him mark the cross for his candidate in Prêt-à-voter. In Demos, the coercer decides which vote-code to vote for. The coercer can then ask voter V_2 to vote at the polling station and vote with this exact ballot $s_1 = (BN_1, s_1^A, s_1^B)$. Since the ballot number is unique for each ballot, the adversary can be sure that the vote was cast as he wanted when he sees the receipt. This attack can be repeated so that V_2 also walks out with an unused ballot $s_2 = (BN_2, s_2^A, s_2^B)$ and let V_3 vote with this particular ballot, and so forth. We note that this attack permits the voters to sell their votes.

Solution A possible solution would be to make it impossible in both system to walk out with an unused ballot. For example, only one is distributed per person and election authority have to check that s^A is destroyed in Prêt-à-voter and exactly one s^i where $i \in \{A, B\}$ is destroyed in Demos.

Picture attack The voter is asked to vote for a particular candidate and take a picture of or film the entire ballot $s = (BN, s^A, s^B)$ while in the polling booth. By comparing the receipt and the unique BN to the picture or film, the adversary can be sure that the vote was cast as he wanted. This is a bigger attack than with the traditional voting system, because a person would have to film every movement in order to show and prove his/her way of voting. This attack permits voters to sell their votes.

Solution A solution is rather difficult to find since the ballot number BN, unique for each vote, permits voter verifiability. Removing it from the system is therefore not possible.

Conclusion Both Prêt-à-voter and Demos show weaknesses regarding the privacy requirement. If these systems do not verify that the voting part of the ballots is destroyed after being used, they are not coercion-resistant. But how can the election authority ensure that people destroy part of their ballots and at the same time ensure privacy in the polling booth? We note also that these attacks are even more critical if the systems are used as remote voting systems. It is extremely difficult to define coercion-resistance and privacy in general, and even more difficult to find a voting system satisfying a coercion resistance definition without sacrificing the verifiability or availability of the system. Although these systems can still satisfy the definition of privacy under the privacy game we see that voters can be coerced during the voting phase.

Chapter 8

Closing Remarks

The primary goal of this thesis was to investigate the potential of paper-based electronic voting systems. We chose to learn more about Demos and Prêt-à-voter, both appealing because they introduce verifiability.

Firstly, we wanted to understand how such systems can be constructed. The investigations required understanding in cryptography, knowledge of voting systems and awareness of the human parameter. Our own presentation of Demos and Prêt-à-voter together with concrete examples was one of the main contribution of this thesis.

Secondly, the goal was to understand what is required of a voting system and motivate the reader to understand how making a voting system trustworthy is complex. To begin with, we analysed what is actually the voter verifiability and how it can influence an election. We demonstrated that this property in Prêt-à-voter opens the possibility of numerous attacks against the privacy. We showed that a voter must be able to verify ballots during the voting phase. Moreover, we concluded that the re-encryption mix-net can be better suited in an election than decryption mix-net because of the lapse of time in between printing and voting. Then, we made an informal analysis of the main cryptographic components of Prêt-à-voter. We implemented RSA cryptosystem in the description of Prêt-à-voter but did not analyse it as many other schemes can be used. We concentrated instead on understanding mix-net, randomised partial checking, length of ciphertext and shuffle proofs. We demonstrated some privacy, verifiability and availability weaknesses and proposed if possible, solutions. A shuffle proof should be used instead of RPC. We proposed to use a padding method to hide the length of ciphertexts and to implement statistical analysis on randomness of the order of the candidate list. At last, we defined mathematically the notion of privacy and explained different attacks against this property in Prêt-à-voter and Demos.

Our goal was quite ambitious as we wanted to obtain an overview of a voting system from beginning to end. If there had been more hours in a day, we would also have implemented different cryptographic schemes in Prêt-à-voter, analysed these and compared the results. We could also have presented a more advanced version of a shuffle proof.

Finally, a personal goal was to get a deeper understanding in this field and to obtain an overall idea of how cryptography can be used in a real life situation. We also wanted to investigate the relation between the mathematics and the human parameter as both must be taken into consideration in an election. We wanted to acquire the knowledge required to follow ongoing research in electronic voting system.

Bibliography

[1] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In *Advances in CryptologyEUROCRYPT 2002*, pages 83–107. Springer, 2002.

[2] Michael R Clarkson Stephen Chong Andrew and C Myers. Civitas: A secure remote voting system. 2007.

[3] David Bismark, James Heather, Roger Peel, Steve Schneider, Zhe Xia, and Peter YA Ryan. Experiences gained from the first pret a voter implementation. In *Requirements Engineering for e-Voting Systems (RE-VOTE), 2009 First International Workshop on*, pages 19–28. IEEE, 2010.

[4] David Chaum, Peter YA Ryan, and Steve Schneider. *A practical voter-verifiable election scheme*. Springer, 2005.

[5] Nikos Chondros, Alex Delis, Dina Gavatha, Aggelos Kiayias, Charalampos Koutalakis, Ilias Nicolacopoulos, Lampros Paschos, Mema Roussopoulou, Giorge Sotirelis, Panos Stathopoulos, et al. Electronic voting systems–from theory to implementation. In *E-Democracy, Security, Privacy and Trust in a Digital World*, pages 113–122. Springer, 2014.

[6] Michael R Clarkson, Stephen Chong, and Andrew C Myers. Civitas: A secure voting system. Technical report, Cornell University, 2007.

[7] Stephanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Computer Security Foundations Workshop, 2006. 19th IEEE*, pages 12–pp. IEEE, 2006.

[8] Alex Delis, Konstantina Gavatha, Aggelos Kiayias, Charalampos Koutalakis, Elias Nikolakopoulos, Lampros Paschos, Mema Rousopoulou, Georgios Sotirellis, Panos Stathopoulos, Pavlos Vasilopoulos, et al. Pressing the button for european elections.

[9] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 61–70. ACM, 2005.

[10] Shahram Khazaei and Douglas Wikström. Randomized partial checking revisited. In *Topics in Cryptology–CT-RSA 2013*, pages 115–128. Springer, 2013.

[11] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-end verifiable elections in the standard model. In *Advances in Cryptology-EUROCRYPT 2015*, pages 468–498. Springer, 2015.

[12] C Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125. ACM, 2001.

[13] Peter YA Ryan. Prêt à voter with paillier encryption. *Mathematical and Computer Modelling*, 48(9):1646–1662, 2008.

[14] Peter YA Ryan, David Bismark, James A Heather, Steve A Schneider, and Zhe Xia. The prêt à voter verifiable election system. *IEEE transactions on information forensics and security*, 4(4):662–673, 2009.

[15] Peter YA Ryan and Steve A Schneider. *Prêt à voter with re-encryption mixes*. Springer, 2006.

[16] Frederic Charles Schaffer. *Elections for sale: the causes and consequences of vote buying*. Lynne Rienner Publishers Boulder, CO, 2007.

[17] D.R. Stinson. *Cryptography: Theory and Practice, Third Edition*. Discrete Mathematics and Its Applications. Taylor & Francis, 2005.