**NTNU – Trondheim**
Norwegian University of
Science and Technology

# Cryptographic Analysis of Voting Systems

Olav Dahl Solstad

Master of Science in Mathematics
Submission date: December 2013
Supervisor: Kristian Gjøsteen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

# Abstract

Coercion-resistance is defined and used to analyze/discuss the voting systems Civitas and the Norwegian Voting Protocol. Prêt à Voter is described both in a real and an ideal case, and analysis of these cases regarding coercion-resistance is done.



# Sammendrag

Coercion-resistance blir definert og brukt til å analysere/diskutere valgsystemene Civitas og den Norske Valgprotokollen. Prêt à Voter blir beskrevet i et ekte og et ideelt tilfelle, og analyse av disse tilfellene m.h.t. coercion-resistance blir gjort.

# Contents

# Chapter 1

# Introduction

## 1.1 The problem

A voting system is a method for allowing voters to submit ballots and then count those ballots to arrive at a result. In modern society voting systems are an important tool with the most important use being political elections, but may be used for less important tasks as well. Traditional paper based voting systems have been used for a long time, most of them requiring a voter to show up at a polling station. In later years electronic voting systems have been used, and with these new systems being deployed there are alot of challenges. Voting systems can be divided into two groups, non-remote systems requires voters to show up at a polling station while remote systems allow the voter to vote remotely.

It is important that voters trust the voting system, so one needs to verify the integrity of an election to some degree. Voting systems usually have different phases, commonly the setup phase, the voting phase and the tallying phase. A verifiable voting system aims to allow voters, election officials and even observers to verify that the different stages of the voting system has been executed correctly. Thus all the stages/steps of a voting system needs to be verifiable by some method for a voting system to be considered verifiable.

There are different ways of achieving verifiability for voting systems. Physical observation is one method of verifying a step, an example being voters observing their ballot being inserted into the ballot box. For other systems physical observation may no longer be enough, and then one needs to generate proofs that can be verified to show that protocols have been followed honestly. Many systems use receipts to enable voters to verify their vote after leaving the polling station or while they are still inside.

In many elections there are people who have something to gain by a cer-

tain election result. These people might use coercion to increase the chances of getting this result, leading to the need of voting systems that can offer some resistance to coercion.

Voters can be divided into two groups of voters, those who are compliant to coercion and those who are non compliant to coercion and will try to resist it by using some strategy. There are many ways for a coercer to get information to help him decide if a voter is compliant or non-compliant with coercion. Voters may be observed at polling stations, and the equipment and people at a polling station may be corrupted. If voters get receipts after voting the coercer may demand that he get the receipt. A coercer may listen in on the voters devices, so using computers, phones and other devices might leak information to the coercer. All the public information that can be gathered from websites, bulletin boards and other sources are also available for the coercer.

Certain attacks does not allow voters to do anything but follow commands, hence the adversary need not distinguish between compliant and non-compliant voters. Forcing voters to abstain or vote in a random fashion are examples of these attacks, and there are many more.

There are many different types of coercers. Some will do very little to ensure that voters do as instructed, while some will use a lot of resources to ensure that voters have no choice but to do as instructed. It is important to model different types of coercers and try to figure out what strength a coercer needs to break coercion-resistance. For some voting systems a weak form of coercion-resistance may be enough, but in other voting systems a stronger form of coercion resistance might be needed.

For voting systems to be used for important tasks it would be ideal to have both verifiability and coercion-resistance. This is hard or maybe impossible to achieve while still having a voting system that is practical to use, so many systems have some degree of coercion resistance as well as some verifiability. The focus in this thesis is on the coercion-resistance of voting systems, with verifiability being mentioned briefly.

## 1.2 Our contribution

A model describing how coercion happens is presented. This model describes voters and a functionality allowing an adversary to listen to a voters communication with devices. Using this model several adversaries are defined, and coercion-resistance for remote voting systems is defined for the different adversaries.

The remote voting systems Civitas and the Norwegian Voting Protocol

are described and then analyzed/discussed using the definition of coercion-resistance.

The non-remote voting system Prêt à Voter is described in more detail than the other voting systems. Both a real and an ideal case is presented, and the relation between these two cases are used to analyze the coercion-resistance properties of Prêt à Voter.

## 1.3 Overview of the thesis

In Chapter 2 the background for the master thesis is presented. Voting systems and end-to-end verifiability is discussed. A coercion-resistance definition is mentioned, and a new definition is presented for remote voting systems.

In Chapter 3-5 several voting systems are described and analyzed. The Prêt à Voter voting system [9] is a paper ballot system aiming to offer end-to-end verifiability and some coercion-resistance. Civitas [6] is a remote voting system based on Juels, Catalano and Jakobsson's voting system presented in [4] aiming to be both end-to-end verifiable and coercion-resistant. The Norwegian Voting Protocol [3] is a remote voting system that provides coercion-resistance, but not end-to-end verifiability. Although the system enables remote voting by using a computer, voting at a polling station is also possible.

Prêt à Voter is described in more detail than the others, and the analysis is done in a different way than using the definitions in Chapter 2. Civitas and the Norwegian Voting Protocol are described with less detail and are analyzed/discussed using the definitions in Chapter 2.

In Chapter 6 the conclusions drawn in this master thesis are summarised.

## 1.4 Acknowledgements

First of all I would like to thank my supervisor Kristian Gjøsteen for all the great advice, guidance and encouragement that has kept me going.

My family and friends have been a great support for me during my years at the university, thank you all!

# Chapter 2

# Background

## 2.1 Voting System

A voting system is a system that allows voters to choose between options by casting a vote for one or more options. After the voting phase is over cast votes are counted to produce a tally, the result of the votes cast. The applications for voting systems are many, but the most important use is political elections where a big percentage of a countrys population votes.

Until recently voting systems have been non-remote, requiring voters to show up at a polling station to cast their vote. Some systems require voters to mark paper ballots which are then cast into a ballot box, while others requires voters to interact with electronic machines to vote.

Remote voting systems does not require voters to show up at a polling station, and could be helpful for many voters who have trouble getting to the polling station when it is open. Some systems allow voting by mail, and others use computers and/or phones. There are also systems that allow voters to vote both remote and non-remote.

In many voting systems one wants the choices a voter made when voting to be secret and/or anonymous. The reason for this might be the laws of countries where the voting system is to be used, or to remove some of the threat of coercion. Another good reason is political privacy, which is an important concern in political elections.

## 2.2 End-to-end verifiable voting systems

End-to-end verifiable (E2E) voting systems are designed in such a way that each stage of an election is verifiable by some method. A simplified description of this is that voters will want to know that they have cast a vote as they

intended, one wants to verify that cast votes are recorded without alteration, and also that the votes are counted as they were recorded.

In some voting systems physical observation is enough to provide some verifiability, but in others one needs more advanced techniques. Including proofs with the outputs of different protocols of a voting system makes auditors able to verify that the protocols have been executed correctly.

E2E voting systems often use receipts that voters can use to help verify an election. Some voters will throw their receipt away the second they have voted, but in big elections it should be possible to convince many voters to help verify the election by using their receipts. One can even make audit teams that take voters receipts and audits them for the voters.

## 2.3 Coercion-resistance

In an election there is a possibility that voters will be coerced into doing something they would not normally have done, which in turn might change the outcome of the election. Coercion might happen when a married couple votes, or it might happen in a more extreme form where voters are threathened with violence.

Coercion-resistant voting systems attempts to remove the threat of coercion, so that voters can vote honestly without fear of being caught by the coercer. To achieve this it must be impossible (or at least very hard) for the coercer to determine what a voter has voted, even if the voter tries to prove what he voted to the coercer.

The basic idea of a coercion-resistant voting system by Juels, Catalano and Jakobsson [4] is: *Hence 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.*

In [4] a new function, *fakekey*, is added to the voting system used in defining coercion-resistance. This function produces a fake key, which is used to resist coercion. The idea behind proving coercion-resistance of a voting system is that when a voter is coerced he will flip a coin, and depending on the coin flip will use either a real key or the fake key to vote. After the election tally is done the adversary wins if he can determine the outcome of the coin flip. This is all defined in an experiment, both in a ”real” setting and an ideal setting, and the voting system is coercion-resistant if the advantage of the adversary is neglible. Advantage is calculated as the absolute difference between the success probabilities of the adversary in the experiments.

This definition is good for certain voting systems, but not very general.

Not all voting systems deploys fake keys, and fake keys might not be particularly good for remote voting systems, something that will be clear when analyzing Civitas which is based on the encryption scheme used in Juels, Catalano and Jakobsson's article [4]. The definition could be altered to be more general, but instead of that a new definition of coercion-resistance is presented. The first thing one needs to look at is the model of coercion, and then we move on to defining coercion-resistance.

### 2.3.1 Model for coercion

The voter is denoted $V$ throughout this master thesis, and the enviroment, $\mathcal{Z}$, used in some of the system specifications is the enviroment of the voter, including the voter itself. The adversary, or coercer, is denoted by $\mathcal{A}$. Voters can be divided into two groups, those who are compliant with coercion and those who are non-compliant with coercion.

The following model of a voter is the one presented by Kristian Gjøsteen at a workshop in Luxembourg [2], and will be important for the analysis of voting systems: The voter is considered as a very limited CPU with unreliable internal memory $M_I$ and reliable external memory $M_E$. Recall from external memory is perfect, while recall from internal memory is imperfect with the fault likelihood increasing with the size of secrets stored there. The coercer may erase the external memory, and the voter cannot lie about the contents of his external memory or his communication with devices. On the other hand, the voter may lie about the content of his internal memory.

The Phys-com functionality allows the voters of a voting system to communicate with devices. When a protocol specifies that something should be sent to a voter or a device running a protocol, this communication will go through the Phys-com functionality. When communication happens the functionality checks to see if either the sender or the reciever is coerced, and if so the adversary gets the contents of the communication. Should a device be considered safe it is easy to add an exception or make a new functionality for that device. Presented underneath is the basic Phys-com functionality describing a weak adversary. $\mathcal{X}$ and $\mathcal{Y}$ are different players/devices that are part of the system.

To model a stronger adversary, that intuively has installed hidden software on the voters devices so he can listen on the voter after releasing him, we can modify the Phys-com functionality so that if the voter is marked as coerced he will never be unmarked. In other words, Phys-com will keep sending messages to $\mathcal{A}$ even if the voter himself thinks he is released from coercion.

On (coerce, $V$, msg) from $\mathcal{A}$:

1. Mark $V$ as coerced.
2. Deliver (coerce, msg) to $V$.

On (release, $V$, msg) from $\mathcal{A}$:

1. Unmark $V$ as coerced.
2. Deliver (release, msg) to $V$.

On (send, $\mathcal{X}$, msg) from $\mathcal{Y}$:

1. If $\mathcal{X}$ or $\mathcal{Y}$ is coerced by $\mathcal{A}$, send (send, $\mathcal{X}$, msg, $\mathcal{Y}$) to $\mathcal{A}$.
2. Deliver (recv, $\mathcal{Y}$, msg) to $\mathcal{X}$.

Figure 2.1: Phys-com functionality for a weak adversary

On (release, $V$, msg) from $\mathcal{A}$:

1. Deliver (release, msg) to $V$.

Figure 2.2: Phys-com modification for a strong adversary

The thought behind the definition of Phys-com is that it is clear whether a voter is in a state of coercion or a state of being released from coercion. Therefore we need to explore strategies for when a voter is coerced (coercion-strategies) and strategies for when a voter is released (recovery-strategies).

Compliant voters will follow the adversary's instructions when coerced, so the adversary completely determines the coercion strategy of these voters. A recovery-strategy is not needed for compliant voters, they are done when they are released. One could argue that the adversary could instruct the voter to do something after being released, but generally the adversary is stronger if all instructions are to be carried out while the voter is under coercion.

Non-compliant voters on the other hand must deploy a coercion-strategy and a recovery-strategy, which in the adversary's view is indistinguishable from the strategies of a compliant voter. These strategies must enable the voter to resist coercion in such a way that he can vote honestly without fear of being caught by the adversary.

The adversary has access to all public information (election results, ...), information gained by listening to voters communication as well as information gained by corrupting certain election agents.

### 2.3.2 Definitions for remote voting systems

The strategies for compliant voters must be simulated by non-compliant voters, so one must look at the worst-case scenario. The coercion-strategy for compliant voters will be to hand over all secrets to the adversary, as well as doing exactly what the adversary instructs. An adversary may instruct the voter to do a lot of things, but essentially the instructions are for a voter to vote for a specified candidate, to vote in a special way or to abstain. The recovery-strategy is empty for compliant voters as they will do what the adversary instructs and then be done. Hence, we get the following definition:

**Definition 2.1.** *The worst-case strategies are:*

- $coerce_{def}$*: hand over all secrets to* $\mathcal{A}$*, vote as* $\mathcal{A}$ *has instructed.*
- $recovery_{def}$*: do nothing.*

Now, if a non-compliant voter could simulate these worst-case strategies he could simulate any "weaker" strategy as well. Hence the adversary would not be able to determine whether a voter is compliant or non-compliant if the simulation is successfull. To determine if this is possible it is necessary to properly define the adversary. As mentioned, the adversary has access to all public information, as well as any information gained through coercion of a

voter. The Phys-com-functionality allows for a weak and a strong adversary, so we get two definitions of the adversary:

**Definition 2.2.** *A weak adversary has access to all public information, coerced voters external memory and information gained by the unmodified Phys-com-functionality.*

**Definition 2.3.** *A strong adversary has access to all public information, coerced voters external memory and information gained by the modified Phys-com-functionality.*

The adversary may also corrupt election agents, so one needs to define a mixed adversary as well although the focus of this master thesis is on the weak and strong adversaries.

**Definition 2.4.** *A mixed adversary is either a weak or a strong adversary who gets additional information through corrupted election agents.*

Coerced voters should be able to vote for their preferred candidate(s) (denoted $\vec{m}$), and if they haven't decided what to vote yet should submit a blank ($\vec{0}$) or random vote ($\vec{m}_{rnd}$). If a voter has submitted a blank vote under coercion he has at the very least partially resisted coercion. If revotes are allowed a voter can later revote for his preferred candidate(s), but only if it is safe to revote. Some voting systems intends for the voter to submit a fake vote when coerced, something which also should be allowed provided it is safe.

Now we can define coercion-resistance using the worst-case scenarios and adversaries defined above, and also keeping in mind that voters should be able to vote honestly.

**Definition 2.5.** *A voting system is coercion-resistant w.r.t. an adversary $\mathcal{A}$ if:*

- *There exists strategies $coerce_{resist}$ and $recovery_{resist}$ that upon execution are indistinguishable from $coerce_{def}$ and $recovery_{def}$ in $\mathcal{A}$'s view, and:*
- *The strategies $coerce_{resist}$ and $recovery_{resist}$ allows a coerced voter to submit a vote for $\vec{m}$ if remembered, and otherwise to vote $\vec{0}$ or $\vec{m}_{rnd}$. Alternatively the strategies allows a coerced voter to submit a fake vote if allowed by the voting system.*

It is natural to classify systems as weak coercion-resistant systems and strong coercion-resistant depending on which adversary is used. Mixed adversaries may be described in many ways, so here it is not natural to divide

into classes. Of course if the system is not weak/strong coercion-resistant it is not mixed coercion-resistant w.r.t. a weak/strong adversary either.

Voting systems that claim to be coercion-resistant comes with one coercion-strategy and/or one recovery-strategy. With coercion-resistance defined as above one can check if the claim is true by testing coercion-resistance for different kinds of adversaries.

### 2.3.3 Non-remote voting system

The definitions used in defining coercion-resistance for remote voting systems could be altered slightly to make definitions for non-remote voting systems as well. For the one non-remote voting system in this master thesis (Prêt à Voter) a different approach was chosen, building a simulator and a functionality that together describe the security against coercion.

# Chapter 3

# Prêt à Voter

## 3.1 Overview

The Prêt à Voter voting system [9] is a paper ballot system aiming to offer end-to-end verifiability and some coercion-resistance. Ballots are made of two separable parts, the left part contains a candidate list in random order and the right part contains boxes for marking as well as some encrypted information that is used to reconstruct the candidate list for tallying purposes. When a ballot is marked and is to be submitted the left part is destroyed and the right part is scanned, signed and uploaded to an append-only bulletin board. Below is an example showing how unmarked ballots, marked ballots and receipts look like.

| | |
|---|---|
| George | |
| Raymond | |
| Richard | |
| Martin | |
| | a8sf7as9 |

| | |
|---|---|
| George | |
| Raymond | |
| Richard | x |
| Martin | |
| | a8sf7as9 |

| |
|---|
| |
| |
| x |
| |
| a8sf7as9 |

A voter marks his ballot by looking at the left part to find his preferred candidate, and then marking the corresponding box on the right part of the ballot. When the ballot is marked the left part is destroyed and the right part is put into a scanner. The boxes of the right part is scanned to identify which box is marked, also the encrypted information on the ballot is scanned. The scanned information is signed and then uploaded to the bulletin board, the marked ballot is given to the voter as a receipt. When the voter wants to check if his vote has been uploaded correctly he can look for his receipt on the bulletin board to check if it appears unaltered. When the election

is over all the contents of the bulletin board are transformed from a set of encrypted votes into a set of unencrypted votes.

The first version of Prêt à Voter was made by Peter Y.A. Ryan. Later an enhanced version was published in [1]. The use of re-encryption mixes was introduced in [8], and also chain voting attacks and chain of custody problems are looked at. In [8] ElGamal was used, and as a result one needs to do extra work to ensure that "the construction of the ballot forms mesh with the re-encryption mixes" [7]. Using the homomorphic properties of Pallier encryption [7] shows how this extra work can be avoided.

Prêt à Voter is designed to be end-to-end verifiable. The voter can check his receipt against the bulletin board, mixnets should include proofs for correct mixing which allows for public verifiability, and some versions of Prêt à Voter even includes auditing at the polling station.

When a voter is coerced to vote for a specific candidate the strategy for resisting coercion is for the voter to vote for his intended candidate, and if the voter has no intended candidate the strategy is to vote randomly. This strategy relies heavily on that the adversary cannot decrypt the encrypted information on receipts, or in any other way reconstruct the candidate list for a receipt. Hence it is very important to have a trusted chain of custody from ballot creation to ballots being used in an election.

There are several simplifications in this chapters description of Prêt à Voter. It is assumed that the scanner and the election authority is honest, limiting the adversary's power. Ballot generation has been simplified by using a public key cryptography functionality which makes the adversary unable to learn anything about a ballots candidate list unless he sees the whole ballot. Tallying is also simplified, so instead of being done in a threshold way the election authority can decrypt votes using the public key cryptography functionality.

The system is modeled by the functionalities BB, $F_{pkc}$ and $F$ as well as protocols for the voter $V$, the computer $P$, the scanner $S$ and the election authority $EA$. Below are short summaries of what each functionality and protocol does. The left and right part of a ballot is denoted $B_l$ and $B_r$ respectively.

## 3.2 Functionalities

The BB functionality models the bulletin board, which is append only. BB recieves and remembers marked ballots from the scanner in the voting phase, and in the tallying phase offers to show all remembered votes to a computer or the $EA$.

The $F_{pkc}$ functionality is a public key cryptography functionality which allows a player to generate his own set of keys/algorithms. When this generation is done any other player can encrypt with a public key, but only the player who ran the key generation can decrypt.

$F$ models the physical actions at the polling station. In the setup phase it recieves ballots from the $EA$ and remembers them. When the voting phase is active the functionality allows voters to fetch, mark and spoil ballots, as well as giving $B_r$ to the scanner and initiating the submission of a ballot. Sending a marked $B_r$ to another player is allowed. The optional part of the functionality allows voters to send complete, unmarked ballots to other players.

*Voting phase*

On ($B_r$, $v$) from $S$:

1. Remember ($B_r$, $v$). Answer (ok).

*Tallying phase*

On (empty) from $P$ or $EA$:

1. Answer {($B_r$, $v$) remembered}.

Figure 3.1: Bulletin board functionality BB

*Setup phase*

On (keygen) from $\mathcal{X}$:

1. Let $(ek, dk) \leftarrow \mathcal{K}$.
2. Remember $(ek, dk, \mathcal{X})$.
3. Answer (ok).

On (enc, $\mathcal{X}$, $m$) from $\mathcal{Y}$:

1. If $(ek, dk, \mathcal{X})$ is remembered let $c \leftarrow ek(0^{|m|})$. Remember $(c, \mathcal{X}, m)$. Answer $c$.

*Tallying phase*

On $(dk, c)$ from $\mathcal{X}$:

1. If $(c, \mathcal{X}, m)$ is remembered, answer $(m)$ and stop.
2. Let $m \leftarrow dk(c)$ and remember $(c, \mathcal{X}, m)$. Answer $(m)$.

Figure 3.2: Ballot encryption functionality $F_{pkc}$

*Setup phase*

On (create, $B_l$, $B_r$) from $EA$:

1. Remember that $B_l$ and $B_r$ are created and that ($B_l$, $B_r$) is a ballot.

*Voting phase*

On (fetch) from $V$:

1. If $V$ has voted, answer (voted) and stop.
2. Register $V$ and select a random ballot ($B_l$, $B_r$) that no other voter has.
3. Remember that $V$ has $B_l$ and $B_r$. Answer (ballot, $B_l$, $B_r$).

On (mark, $B_r$, $v$) from $V$:

1. If $V$ does not have $B_r$, stop and answer (hasnot).
2. If $(B_r, v')$ is remembered and $v \neq v'$, remember (spoil, $B_r$), forget ($B_r$, $v'$), answer (spoil) and stop.
3. If $(B_r, v)$ is not remembered, remember $(B_r, v)$ and answer (ok).

On (spoil, $B_r$) from $V$:

1. If $V$ does not have $B_r$, stop and answer (hasnot).
2. If ($B_r$, $v$) is remembered, forget ($B_r$, $v$).
3. Remember (spoil, $B_r$) and answer (spoiled).

On (give, $B_l$, $B_r$, $S$) from $V$:

1. If $V$ has voted, answer (voted) and stop.
2. If $V$ does not have $B_l$ and $B_r$, answer (hasnot) and stop.
3. If (given, $B_l$, $B_r$) is remembered, stop.
4. Send ($B_r$, info) to $S$ where info is any information remembered about $B_r$ and wait for a response. If (spoil, $B_r$) is recieved, destroy $B_l$ and $B_r$, send (spoil) to $V$ and stop. If (blank, $B_r$) is recieved send (blank, $B_l$, $B_r$) to $V$and stop. If (ok) is recieved send (ok) to $V$, remember that $S$ has $B_r$, that $V$ had $B_r$ and remember (given, $B_l$, $B_r$).

Figure 3.3a: System functionality $F$

On (submit, $B_l$, $B_r$) from $V$:

1. If $B_l$ or $B_r$ have been submitted, answer (submitted) and stop.
2. If $B_r$ is not in $S$, answer (hasnot) and stop.
3. If $V$ did not have $B_r$, answer (hadnot) and stop.
4. Send (submit, $B_r$, $v$) to $S$ and wait for (ok).
5. Destroy $B_l$, give ($B_r$, $v$) to $V$ as a receipt and remember that $V$ has $B_r$. Remember that $S$ no longer has $B_r$.
6. Remember that $B_l$ and $B_r$ are submitted and that $V$ has voted.

On (send, $B_r$, $v$, $V'$) from $V$:

1. If $V$ does not have $B_r$ stop. Give ($B_r$, $v$) to $V'$ and remember that $V'$ has $B_r$.

Figure 3.3b: System functionality $F$

*Voting phase*

On (send, $B_l$, $B_r$, $V'$) from $V$:

1. If $V$ does not have or have submitted $B_l$ and $B_r$ stop. Give $B_l$ and $B_r$ to $V'$ and remember that $V'$ has $B_l$ and $B_r$.

Figure 3.4: Optional part of $F$ for chain voting

## 3.3 Protocol

The voter's protocol allows a voters intention to be remembered, and contains a description of how a normal voting session is done. The protocol describes how voters should handle coercion by using ballot identifiers when communicating with the adversary, as well as the strategy for resisting coercion. A simple description of how auditing is done is also included. Note that interpreting $B_l$ as a function $f$ and applying it on a candidate $v$ is denoted $B_l(v)$ for simplicity.

The computers protocol is very simple as all it is used for is auditing, namely it checks if a voters receipt is in the collection of remembered votes obtained from BB.

The scanners protocol have two events. It recieves $B_r$ together with information $F$ remembers about $B_r$ and then holds on to $B_r$ if it is marked correctly, otherwise it answers with a message about why $B_r$ was not accepted. When a voter wants to submit a ballot which is in the scanner, the scanner recieves a submission request from $F$ and then sends the marked ballot to BB.

The $EA$'s protocol describes how ballot generation and tallying is done. $B_l$ is the output of a function $f : O \to O$ where $O = \{0, 1, ..., L-1\}$ and L is the number of candidates, and $B_r$ has the information ($c$, obtained through $F_{pkc}$) needed to find this function. When marked $B_r$ consists of $(v, c)$. The $EA$ will use $F_{pkc}$ to decrypt $c$ to find $f$, and then $f^{-1}(v) = v'$. Note that candidates are represented as integers.

*Voting phase*

On input (intention, $v$):

1. If (intention, $v'$) is remembered, forget (intention, $v'$). Remember (intention, $v$).

On input (vote, $v'$):

1. Stop if already voting or if $V$ is coerced. Otherwise mark as voting.
2. If an error occurs in the following steps, stop.
3. Send (fetch) to $F$ and wait for (ballot, $B_l$, $B_r$).
4. Compute $v \leftarrow B_l(v')$.
5. Send (mark, $B_r$, $v$) to $F$ and wait for (ok).
6. Send (give, $B_l$, $B_r$, $S$) to $F$ and wait for (ok). Send (submit, $B_l$, $B_r$) to $F$ and wait for ($B_r$, $v$).
7. Mark as not voting.

On (coerce) from Phys-com:

1. Stop if $V$ is marked as voting. Otherwise, remember that you are coerced and answer (ok).

On (do, msg) from Phys-com:

1. If not coerced, stop.
2. Interpret msg as a command to run, and run it with the following exceptions:

   If (fetch) is a command, run it and when getting (ballot, $B_l$, $B_r$) remember $i = (B_l, B_r)$. When getting commands exchange $i$ with $B_l$ and $B_r$ as needed for the specific command.

   If (msg) is interpreted as the marking of a ballot for $v'$ do the following:

   If (intention, $v$) is remembered compute $v_f \leftarrow B_l(v)$ and run (mark, $B_r$, $v_f$). Otherwise choose a random candidate $v_{rnd}$ and run (mark, $B_r$, $v_{rnd}$).
3. When getting a response (answer) from $F$, send (send, $\mathcal{A}$, answer) to Phys-com with the following exception:

   Whenever (answer) includes $B_l$ or $B_r$ use $i$ to reference them instead of using the actual ballots.

Figure 3.5a: Voter's protocol

On (release) from Phys-com:

1. Forget that you are coerced. Answer (ok).

*Tallying phase*

On input (audit, $P$):

1. Send (audit, $B_r$, $v$) to $P$ and wait for response. Output response.

Figure 3.5b: Voter's protocol

*Tallying phase*

On input (audit, $B_r$, $v$):

1. Send (empty) to BB and wait for $\{x_1, x_2, \ldots\}$.
2. Search for ($B_r$, $v$) in $\{x_1, x_2, \ldots\}$. If found answer (ok), otherwise answer (error).

Figure 3.5c: Computer protocol

*Voting phase*

On ($B_r$, info) from $F$:

1. Interpret info to see what is remembered about $B_r$.
2. If $B_r$ is blank, answer (blank, $B_r$) and stop.
3. If (spoil, $B_r$) is remembered, answer (spoil, $B_r$) and stop.
4. If ($B_r$, $v$) is remembered, answer (ok) and stop.

On (submit, $B_r$, $v$) from $F$:

1. Send ($B_r$, $v$) to BB and wait for (ok). Send (ok) to $F$.

Figure 3.6: Scanner protocol

*Setup phase*

On input (setup):

1. Send (keygen) to $F_{pkc}$ and wait for (ok).
2. Do the following steps as many times as needed.
3. Choose random bijective function $f : O \to O$.
4. Send (enc, $EA$, f) to $F_{pkc}$ and wait for ($c$).
5. Let $B_l \leftarrow f(O)$ and let $B_r \leftarrow c$.
6. Send (create, $B_l$, $B_r$) to $F$.

*Tallying phase*

On input (count):

1. Send (empty) to BB and wait for $\{x\}$.
2. For each ($B_r$, $v$) in $\{x\}$ do the following:
3. Interpret $B_r$ as $c$, send $(dk, c)$ to $F_{pkc}$ and wait for $f$.
4. Compute $f^{-1}$ and then let $v_c \leftarrow f^{-1}(v)$. Remember $v_c$.
5. When done for all ($B_r$, $v$) output all remembered $v_c$

Figure 3.7: $EA$'s protocol

## 3.4 Analysis

### 3.4.1 The polling station functionality

When describing Prêt à Voter with several functionalities and protocols one gets a detailed description of the voting system. An important question is then whether this description models reality, or what is needed for it to better model reality. To limit this discussion slightly, the focus is on whether $F$ models reality or not.

$F$ models a polling station, handling the physical actions taken inside the polling station. To completely model a polling station would be an enourmous task, as there are many variations between different polling stations. People working in the polling station may not follow protocol, there may be issues with equipment availability, etc. Hence the $F$ functionality does not completely model reality.

As adapting $F$ to fully model reality seems like a close to impossible task one might wonder if $F$ could be used to construct a real polling station. In the setup phase $F$ gets sent ballots just like a real polling station would get ballots, but there are complications here with how and by who the ballots are handled. The same complications are present in the voting phase as well, and the root of these complications is easy to spot. $F$ models the polling station as if it is a single unit, while in reality the polling stations consists of many different people, devices, etc. To use $F$ as a basic guideline for constructing a polling station is possible of course, but each event must be inspected carefully to make sure the polling station as a whole functions somewhat close to $F$.

### 3.4.2 Two attacks

There are viable attacks against Prêt à Voter, and two well-known attacks are the randomization and chain voting attacks.

In a randomization attack the adversary will coerce the voter to mark a specific box without caring which candidate that vote is for. When the voter then shows the adversary his receipt it is easy to check if the voter complied. The functionalities and protocols of Prêt à Voter can easily be modified to support this attack by changing from candidate notation to vector notation as well as making some other changes, or simply let the adversary tell the voter to mark a fixed integer/candidate on the ballot.

Chain voting is an attack that is enabled by the optional part of the $F$ functionality when voters can send ballots to other players. An adversary would need to get his hands on an unmarked ballot to start the chain voting

attack. Then the adversary can coerce voters to vote on a ballot supplied by the adversary and handing over the ballot they get at the polling station to the adversary. This way the adversary sees all the ballots and can check receipts or the bulletin board to ensure that coerced voters complied.

### 3.4.3 Coercion-resistance

To be able to do analysis on Prêt à Voter we need to construct a functionality with the security properties we want, as well as a simulator that will simulate the original system. These are presented below, as well as summaries of how voting and coercion can happen in both the original system and the new one.

*Voting phase*

On (intention, $v$) from $V$:

1. If (intention, $v'$, $V$) is remembered, forget (intention, $v'$, $V$). Remember (intention, $v$, $V$). Send (intention, $V$) to $S_{F'}$ and wait for (ok). Send (ok) to $V$.

On (vote, $v$) from $V$:

1. Stop if $V$ is already voting, has voted or is coerced. Otherwise, mark $V$ as voting.
2. Remember (vote, $v$, $V$). Remember that $V$ has voted.
3. Send (vote, $V$) to $S_{F'}$ and wait for (ok). Send (ok) to $V$.
4. Mark as not voting.

On (coerce, $V$, msg) from $S_{F'}$:

1. If $V$ is marked as voting, send (stop) to $S_{F'}$ and stop. Otherwise send (ok) to $S_{F'}$ and mark $V$ as coerced.

On (vote, $V$) from $S_{F'}$:

1. Stop if $V$ has voted or is not coerced.
2. (intention, $v$, $V$) is remembered, remember (vote, $v$, $V$) and answer ($v$). Remember that $V$ has voted.

On (vote, $V$, rnd) from $S_{F'}$:

1. Stop if $V$ has voted or is not coerced.
2. Pick a random candidate $v_{rnd}$, remember (vote, $v_{rnd}$, $V$) and answer ($v_{rnd}$). Remember that $V$ has voted.

On (release, $V$, msg) from $S_{F'}$:

1. Forget that $V$ is coerced. Answer (ok).

Figure 3.8: Election functionality $F'$

*Voting phase*

On (intention, $V$) from $F'$:

1. Remember (intention, $V$). Answer (ok).

On (vote, $V$) from $F'$:

1. Remember that $V$ has voted. Answer (ok).

On (coerce, $V$, msg) from $\mathcal{A}$:

1. Send (coerce, $V$, msg) to $F'$ and wait for a response. If (ok) is responded mark $V$ as coerced and send (ok) to $\mathcal{A}$, otherwise stop.

On (release, $V$, msg) from $\mathcal{A}$:

1. Send (release, $V$, msg) to $F'$. Wait for (ok) and send (ok) to $\mathcal{A}$. Forget that $V$ is coerced.

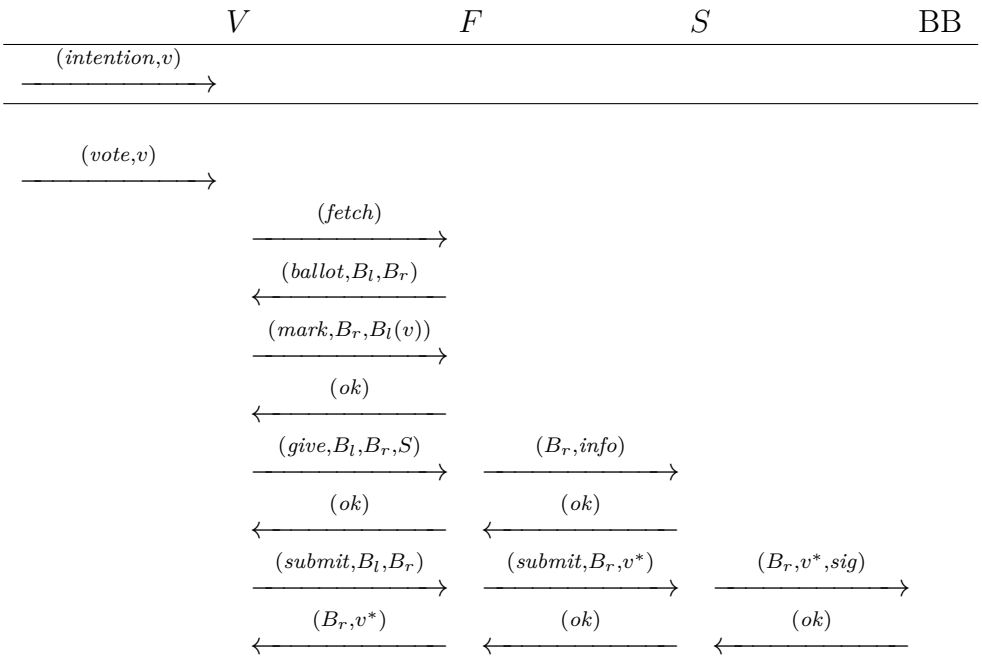
Figure 3.9a: Simulator $S_{F'}$

On (send, $V$, do, msg) from $\mathcal{A}$:

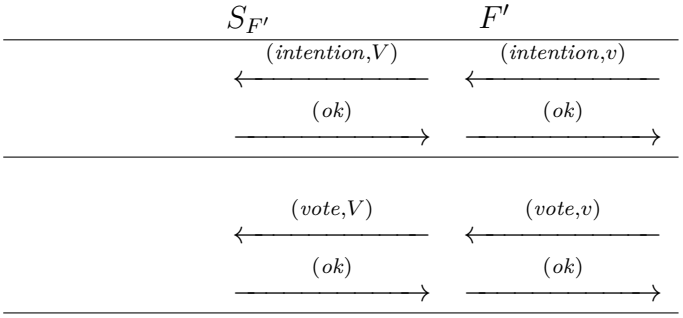
1. Stop if $V$ is not coerced. Interpret msg as a command and follow the instructions for that command:
2. (fetch):
   (a) If $V$ has voted send (voted) to $\mathcal{A}$ and stop.
   (b) Pick an identifier $i$ for a ballot. Send (ballot, $i$) to $\mathcal{A}$. Remember ($V$, i).
3. (mark, i, $v$):
   (a) If ($V$, i) is not remembered, send (hasnot) to $\mathcal{A}$ and stop.
   (b) If (i, $v'$) is remembered and $v \neq v'$, forget (i, $v'$), remember (spoil, i), send (spoil) to $\mathcal{A}$ and stop.
   (c) Remember (i, $v$) and send (ok) to $\mathcal{A}$.
4. (spoil, i):
   (a) If ($V$, i) is not remembered, send (hasnot) to $\mathcal{A}$ and stop.
   (b) Forget any (i, $v$) that is remembered, remember (spoil, i) and send (spoiled) to $\mathcal{A}$.
5. (give, i, $S$):
   (a) If $V$ has voted, send (voted) to $\mathcal{A}$ and stop.
   (b) If ($V$, i) is not remembered, send (hasnot) to $\mathcal{A}$ and stop.
   (c) If (given, i) is remembered stop.
   (d) If (spoil, i) is remembered send (spoil) to $\mathcal{A}$, forget i and stop.
   (e) If no (i, $v$) is remembered send (blank, i) to $\mathcal{A}$ and stop.
   (f) Send (ok) to $\mathcal{A}$ and remember (given, i).
6. (submit, $i$):
   (a) If (submitted, i) is remembered, send (submitted) to $\mathcal{A}$ and stop.
   (b) If (given, i) is not remembered, send (hasnot) to $\mathcal{A}$ and stop.
   (c) If ($V$, i) is not remembered, send (hadnot) to $\mathcal{A}$ and stop.
   (d) If (intention, V) is remembered send (vote, V) to $F'$. Otherwise send (vote, V, rand) to $F'$.
   (e) Wait for a response from $F'$. Send (i, response) to $\mathcal{A}$. Remember (submitted, i) and that $V$ has voted.

Figure 3.9b: Simulator $S_{F'}$

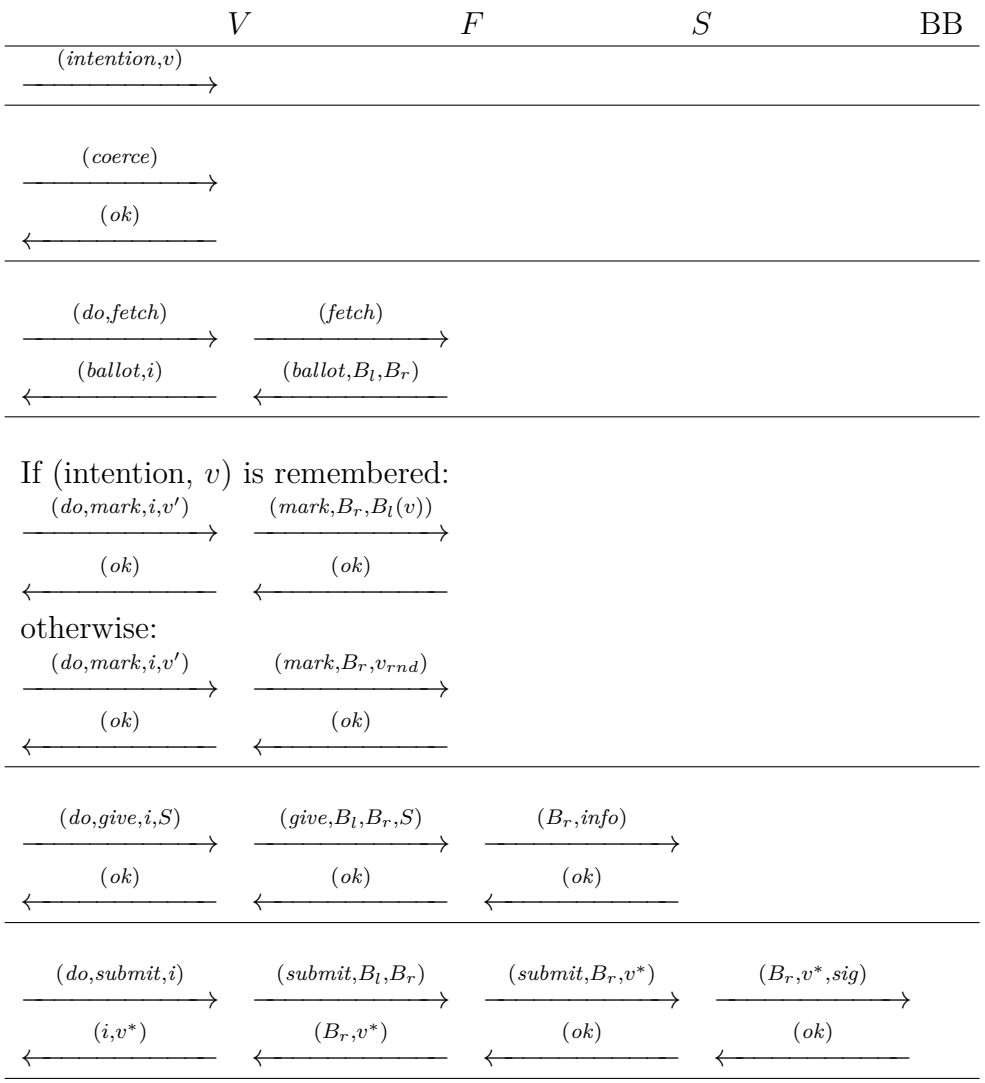
Voting in the real case



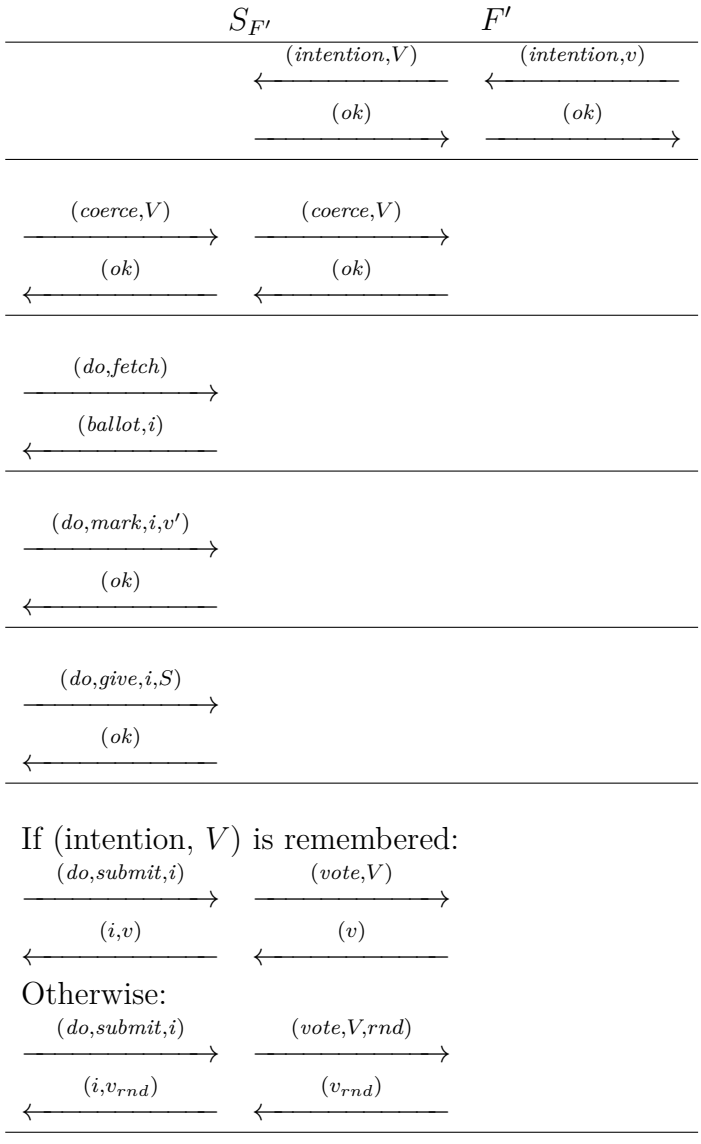

Voting in the ideal case



Coercion in the real case



Coercion in the ideal case



There are two different cases to look at when analysing Prêt à Voter. The real case is described by the functionalities BB, $F_{pkc}$, $F$ and Phys-com as well as protocols for $V$, $P$, $S$ and $EA$. The ideal case is described by a simulator $S_{F'}$ and a functionality $F'$ which only requires inputs (intention, $v$) and (vote, $v$) from the voter. It is easy to inspect both cases and come to the following conclusion: *The real case realizes the ideal case when $F$ does not include the optional part and randomization attacks are ignored.*

Hence the properties of the ideal case are also properties of the real case, and some key properties to the ideal case are:

- If a coerced voter has an intention $v$, he can vote for $v$ even if coerced to vote for some other candidate.
- If a coerced voter has no intention, he can vote randomly if coerced to vote for a specific candidate.

With these properties we draw the following conclusion:

**Conclusion.** *Prêt à Voter described in the real case where the optional part of $F$ is not included is not resistant against randomization attacks. If the attack is not a randomization attack a voter can vote for his intended candidate, and if no such candidate exists the voter can vote for a random candidate.*

There are some things to take note of about this analysis regarding its limitations and future work.

It is assumed that the scanning machine is honest, and dropping this assumption would be of interest. A corrupt scanner would include the adversary more, and might lead to complications regarding coercion-resistance.

The election authority is also assumed to be honest. If this is not assumed there is alot of analysis to be done on the creation and handling of ballots as well as decryption/tallying.

One could give the adversary more power by letting him intercept messages and maybe even alter them. Including the adversary more in the steps of the voting protocol would be interesting to analyze with all the messages that are being sent around in the system.

# Chapter 4

# Civitas

## 4.1 Overview

Civitas [6] is a remote voting system based on Juels, Catalano and Jakobsson's voting system presented in [4] aiming to be both end-to-end verifiable and coercion-resistant. Voters register with multiple registration tellers, and encrypts their vote using a computer which sends the encrypted vote to one or more ballot boxes.

A voter $V$ registers with all registration tellers $RT$ in order to be able to compute a credential. The voter then inputs his vote into a computer which encrypts the vote and sends it to some or all ballot boxes. A zero knowledge proof is also included for verifiability. When tallying begins the tabulation tellers $TT$ gather all the ballots from the ballot boxes and runs a decryption protocol.

Civitas is designed to be end-to-end verifiable. In [6] it is stated *The final tally is verifiably correct. Each voter can check that their own vote is included in the tally (voter verifiability). Anyone can check that all votes cast are counted, that only authorized votes are counted, and that no votes are changed during counting (universal verifiability).*

The coercion-strategy for non-compliant voters is to generate a fake credential to use under coercion. Hence the voter does not have to do anything when being released from coercion, for the real credential has not been used. Of course it is important that the generation of a fake credential does not produce a real credential.

The supervisor of the election $S$ decides whether revoting is allowed and how it is handled. If revotes are not allowed and there are multiple votes with the same credential these votes must be excluded from the tallying. The authors of [6] state that if revoting is allowed voters should include a proof

indicating which earlier vote is to be erased as well as showing knowledge of the choice and credential used in earlier votes.

The description of Civitas in this chapter is not very detailed, the focus is on discussing how the voter can handle coercion. With a more detailed description more analysis could be done, but even with a simple description some analysis can be done. Below are summaries of what each player does.

## 4.2 System description

The supervisor $S$ generates and posts keys for the election in the setup phase. When tallying starts $S$ waits for the ballot box to post commitments to its contents on a bulletin board and then sign these commitments.

The registrar $\mathcal{R}$ publishes voters registration key and designation key.

Registration tellers $RT$ generate private and public credential shares for each voter in the setup phase. When a voter registers an AES session is initiated with the registration teller which then sends his share of the private credential to the voter along with a random factor, a ciphertext and a proof.

The tabulation tellers $TT$ create a public ElGamal key together, and keep a share of the corresponding private key. When tallying starts the votes are retrieved from the ballot box and the public credentials are retrieved from the bulletin board. Proofs are verified for each vote and votes with invalid proofs are removed, duplicates are removed using a plaintext equivalence test. All remaining votes are sent through a mix-network, and then votes with invalid credentials are removed. Then all the remaining votes are decrypted and the tally is published.

The voter $V$ registers by sending his keys to the computer. Voting for a candidate only requires the voter to input the candidate and a credential into the computer. If coerced the voter uses a fake credential and thus submits a fake vote.

The computer $P$ registers with all registration tellers when getting the voters keys. Then the computer computes the voters private credential and a fake credential which are both sent to the voter. When a voter requests that the computer submit a vote for a candidate using a credential the computer encrypts both the credential and the candidate, as well as providing zero knowledge proofs. The encryptions and proofs are then sent to ballot boxes.

The bulletin board $\mathcal{BB}$ is append only and remembers all inputs.

The ballot box $\mathcal{B}$ remember all votes that are sent to it and posts a commitment for all remembered votes on $\mathcal{BB}$.

*Setup phase*

On (setup) from self:

1. Post RSA public keys representing election authorities to $\mathcal{BB}$.
2. Post (p,q,g) describing the ElGamal parameters: $p = 2kq+1$ and a generator g for the order-q subgroup, M, of $Z_p^*$ on $\mathcal{BB}$.
3. Send (setup) to $TT$. Send (setup) to $\mathcal{R}$. Send (setup) to $RT$.

*Tallying phase*

On (close) from self:

1. Send (close) to $\mathcal{B}$. Wait for $\mathcal{B}$ to post commitment on $\mathcal{BB}$. Sign all these commitments. Send (close) to $TT$.

Figure 4.1: The Supervisor

*Setup phase*

On (setup) from $S$:

1. Each $TT_i$ joins in generating a public ElGamal key $K_{TT}$ and keep a share $K_{TT}^i$ for the corresponding private key.

*Tallying phase*

On (close) from $S$:

1. Retrieve votes from each $\mathcal{B}$ and public credentials from $\mathcal{BB}$.
2. Verify proofs of well-formedness for each ballot. Discard ballots with invalid proof.
3. Eliminate duplicates using plaintext equivalence test.
4. Run all remaining votes through a mix-network.
5. Discard votes with invalid credentials.
6. Decrypt votes and publish tally.

Figure 4.2: The Tabulation Tellers

*Setup phase*

On (setup) from $S$:

1. For each $V$ post registration key (RSA) $r_V$ and designation key (ElGamal) $d_V$ to $\mathcal{BB}$.

Figure 4.3: The Registrar

*Setup phase*

On (setup) from $S$:

1. Each $RT_i$ do the following for each $V$: generate a random element of M as a private credential share $s_i$ and post corresponding public share $S_i = Enc(s_i, K_{TT})$ on $\mathcal{BB}$.

   After all $RT_i$ have posted $S_i$ the voters public credential is publicly computable as $\Pi_i S_i$

*Voting phase*

On (register, $r_V$) from $P$

1. Establish a shared AES session key (using Needham-Schroeder-Lowe protocol).

On (share, $d_V$) from $P$:

1. Choose r at random
2. Compute $S'_i = Enc(s_i, r, K_{TT})$ and D, a designated-verifier reencryption proof (requires $d_V$).
3. Send $(s_i, r, S'_i, D)$ to $P$

Figure 4.4: The Registration Tellers

*Voting phase*

On (register) from $\mathcal{Z}$:

1. Send $(r_V, d_V)$ to $P$ and wait for (registered, s, $s_{\mathcal{A}}$). Record $s$ and $s_{\mathcal{A}}$.

On (vote, v, credential) from $\mathcal{Z}$:

1. Stop if already voting. Mark as voting.
2. Send (vote, v, credential) to $P$
3. Mark as not voting.

On (coerce, msg) from Phys-com:

1. Remember that you are coerced.
2. Interpret v from msg.
3. Hand over $s$ and $s_{\mathcal{A}}$ to $\mathcal{A}$.
4. Proceed as if input (vote, v, $s_{\mathcal{A}}$) was recieved from $\mathcal{Z}$.

On (release, msg) from Phys-com:

1. "Forget" that you are coerced.

Figure 4.5: The Voter

*Voting phase*

On (recv, $r_V, d_V$) from $V$:

1. For all $RT_i$ do steps 2-4.
2. Send (register, $r_V$) to $RT_i$ and join in establishing the AES session.
3. Send (share, $d_V$) to $RT_i$ and wait for $(recv, (s_i, r, S_i', D))$
4. Verify that $S_i'$ was computed correctly and that D is correct.
5. When done for all i: construct private credential $s = \Pi_i s_i$
6. Substitute at least one $s_i$ with a random group element, compute and record $s_{\mathcal{A}}$ as $\Pi_i s_i$ with the substituted $s_i$'s.
7. Send (registered, $s$, $s_{\mathcal{A}}$) to $V$

On (vote, v, credential) from $V$

1. Post $\langle Enc(credential, K_{TT}), Enc(v, K_{TT}), P_\omega, P_k \rangle$ to some or all ballot boxes.

   Note: v is the voters choice, $P_\omega, P_k$ are zero knowledge proofs that proves that the vote is well formed and that the submitter knows $s$ and $v$.

Figure 4.6: The Computer

*Voting phase*

On $(c_1, c_2, P_\omega, P_k)$ from $P$:

1. Remember $(c_1, c_2, P_\omega, P_k)$.

*Tallying phase*

On (close) from $S$:

1. Post commitment to all $(c_1, c_2, P_\omega, P_k)$ that are remembered on $\mathcal{BB}$.

Figure 4.7: The Ballot Box

On input (x):

1. Remember x.

Figure 4.8: The Bulletin Board

## 4.3 Analysis

Civitas bases its security on the generation of a fake credential to be used when coerced. The fake credential together with some assumptions are used in [6] to prove Civitas secure under Juels, Catalano and Jakobsson's definition of coercion-resistance [4]. This analysis does not deploy these security assumptions, but does discuss most of them.

### 4.3.1 Externally stored credentials

As most voters would have trouble remembering a real credential and a fake credential let us first assume that both credentials are stored in $M_E$. The secret of which is the real credential is however stored in $M_I$.

The voter might forget which credential is real, and could with 50% probability cast a vote that will not be tallied. In this case, the voter could vote once with each credential and be assured that one of the votes will be tallied.

One could think that generating many fake credentials would stop an adversary, but constructing more than one fake credential does not help. The adversary could just make the voter vote with all credentials and be assured that one of the votes cast is valid. Hence we only generate one fake credential. Now, the adversary can make the voter vote with both credentials and be assured that one of the votes cast is valid. Therefore the recovery-strategy would be to revote, if revotes are allowed at all. This will be safe when dealing with a weak adversary, but a strong adversary will detect the revoting. Hence Civitas is weak coercion-resistant when revoting is allowed, but not strong coercion-resistant even with revoting allowed.

When generating a fake credential one or more shares of registration tellers are swapped before computing the product of the shares. If one of the registration tellers that got his share swapped is corrupted by a mixed adversary, the share will not be part of the fake credential and hence the adversary learns which credential is real.

### 4.3.2 Internally stored credentials

Now consider the case where one stores both credentials in $M_I$.

The coercion-strategy would be to provide the adversary with a fake credential and not the real credential, but the adversary could just ask for both credentials. One could add another fake credential, but then the adversary would just ask for three credentials. Hence the voter must give the adversary the fake credential and either the real credential or a fake one made up by the voter.

A voter might not remember the credentials correctly and will then be unable to vote unless he can "reregister". If registering again is not made possible a majority of the voters might be unable to vote, and the voting scheme is not suitable for most elections. If the voter has been coerced and has given credentials to the adversary, reregistering might not give the same fake credential that was given the first time. Being coerced again the credentials given to $\mathcal{A}$ might then not be the same as the last time the voter was coerced, and the adversary could use this to figure out if the voter is compliant or not.

The adversary could coerce the voter, get two credentials and see the voter cast a vote with one or both of them. If revotes are allowed the voter can vote again, and will not be detected by a weak adversary. Also, if the voter gives the adversary two fake credentials on purpose, he may after being released vote honestly with the real credential. This way a weak adversary does not know whether coercion succeeded. A strong adversary could release the voter and wait to see if another vote is cast. If the credential used is different than the credentials used in the coercion-stage or if revotes are allowed the strong adversary knows the voter is non-compliant. Hence Civitas is weak coercion-resistant when credentials are stored internally, but not strong coercion-resistant.

Should a mixed adversary manage to corrupt a registration teller, the teller would with non-negligble probability not have his share in a made up credential and say that it is fake, but hopefully not be able to say something about the other credential. Should the voter get caught in handing over two fake credentials the only defense would be to blame his memory. Coercion-compliant voters might also remember wrongly, so the adversary can not know whether a voter is compliant or not with absolute certainty unless the voter does something to indicate this later in the election (revoting, voting with different credential, ...).

With this model the ability to vote is severely weakened. Many people have trouble remembering their PIN-codes for phones and credit cards. In an election the credentials are only used once, so one does not even have time to get the numbers "automated" into the voters. Hence, this model is not realistic with the assumption that people can remember two (possibly quite large) credentials that are going to be used only once. The adversary might

also have problems in this model, due to the fact that peoples memory is faulty. If the credentials are hard to remember, coerced voters might give only fake credentials without intending to do so.

### 4.3.3 Security assumptions

Civitas is proven secure with Juels, Catalano and Jakobsson's definition of coercion-resistance by using several trust assumptions, and hence it is necessary to analyze some of the assumptions to see if they are realistic.

- *The adversary cannot simulate a voter during registration. (Trust Assumption 1 in [6])* If the adversary could simulate a voter in both the registration and the voting phase the adversary could just vote on behalf of the voter. One strategy for avoiding this is to use in-person registration, which should prevent the adversary from simulating a voter during registration.

- *Each voter trusts at least one registration teller, and the channel from the voter to the voter's trusted registration teller is untappable. (Trust Assumption 2 in [6])* Voters will generally trust election officials, but rarely will a voter be completely sure that the trust is not misplaced. Untappable channels could be possible in certain registration scenarios, which of course must satisfy the previous assumption as well.

- *Voters trust their voting clients. (Trust Assumption 3 in [6])* Assuming voters can put their trust in their personal computers or some other computer that is accessible for them this is not a totally unrealistic assumption in many scenarios. Most voters does not have great knowledge of how to check if their computer has any infections, and they rely heavily on programs installed by the seller/manifactor of the computer. Old computers with outdated security-programs are easy targets, and even well updated computers may be infected by the more dedicated adversaries. A report by Microsoft [5] shows that if computers have some protection software installed somewhere around 2-4% of computers are infected with malicious software. If the computers are unprotected this percentage increases to around 12-14%. Some countries like Pakistan and Georgia are worse off than others, so one needs to think about which countries will use the voting system as well. If these percentages are acceptable for election authorities it may be assumed that voters can put their trust in their voting clients.

- *The channels on which voters cast their votes are anonymous. (Trust Assumption 4 in [6])* As voting clients aren't necesserily trustworthy this assumption does not necessarily hold as the voting clients are part of the channel. There are also problems with WiFi networks and internet providers that might cause problems for anonymity.

- *At least one of the ballot boxes to which a voter submits his vote is correct. There exists at least one honest tabulation teller. (Trust Assumption 5 and 6 in [6])* It is not unrealistic to assume that most ballot boxes function correctly, and the probability that all ballot boxes a voter submits his vote to are faulty should be small in most settings. That at least one tabulation teller is honest is a very weak assumption which should hold for most elections.

# Chapter 5

# The Norwegian Voting Protocol

## 5.1 Overview

The Norwegian Voting Protocol [3] is a remote voting system that provides coercion-resistance, but not end-to-end verifiability. Although the system enables remote voting by using a computer, voting at a polling station is also possible.

A voter $V$ inputs his ballot into a computer $P$ that encrypts the ballot and sends it to a ballot box $B$. The ballot box cooperates with a return code generator $R$ to generate return codes that are sent by SMS to the voters phone $F$. When recieveing the return codes the voter verifies the codes against a list of option-return code pairs on his voting card. As the ballot box closes all submitted ballots are decrypted by a decryptor $D$. There is also an auditor $A$ that supervises the process.

Ballots and whether a person has voted is kept secret in the norwegian voting protocol. If a voter revotes electronically, all previous electronic votes will not be counted. Should a voter vote physically all electronic votes will not be counted. Voters can vote electronically from a given date before the election until the election day, then only physical voting is allowed.

The version of the voting protocol presented is a simplified version of the cyptographic protocol used in 2011. As the focus of [3] is on the ballot submission protocol the decryption protocol is not specified in the instantation.

A coerced voter will follow the adversary's instructions, and when released from coercion will revote. This will make sure that the vote cast under coercion will not be counted.

This chapter is very heavily based on [3]. The protocol specifications are almost exactly identical, although some small changes have been made. First some algorithms are presented, and then functionalities and protocols.

## 5.2 Algorithms

All (close to) as stated in [3]:

The set of group elements of a group G will be both the message space M and the set of pre-codes C. Interpret O as the set of options, and 1 as the null option and null code. Denote a set of voters $\mathcal{V}$ and a set of computers $\mathcal{P}$.

The set of pre-code maps S is the set of automorphisms on G, which corresponds to the set of exponentiation maps $\{x \to x^s | s \in \{1, 2, ..., q-1\}\}$. Commit to a pre-code map s by computing $s(G)$.

Define the map $\omega$ as $\omega(\vec{m}) = \phi(m_1 m_2 ... m_k)$.

- The key generation algorithm $\mathcal{K}$ samples for each i from 1 to k $a_{1,i}$ and $a_{2,i}$ uniformly at random from $\{0, 1, ..., q-1\}$, then computes $a_{3,i} = a_{1,i} + a_{2,i}$ mod q, $y_{1,i} = g^{a_{1,i}}, y_{2,i} = g^{a_{2,i}}$ and $y_{3,i} = g^{a_{3,i}}$. The public key is

  $$ek = (y_{1,1}, ..., y_{1,k}, y_{2,1}, ..., y_{2,k}, y_{3,1}, y_{3,k})$$

  The decryption key is $dk_1 = a_1 = \Sigma_{i=1}^{k} a_{1,i}$ mod q, the transformation key is $dk_2 = (a_{2,1}, ..., a_{2,k})$ and the code decryption key is $dk_3 = (a_{3,1}, ..., a_{3,k})$.

- The pre-code map generation algorithm $\mathcal{L}(ek, V)$ samples s uniformly from the set $\{1, 2, ..., q-1\}$. It computes $\gamma = g^s$ and outputs the map determined by s and the commitment $\gamma$.

- The encryption algorithm $\mathcal{E}(ek, V, \vec{v})$ samples a random number t uniformly at random from $\{0, 1, 2, ..., q-1\}$, computes $x = g^t$ and $\omega_i = y_{1,i}^t v_i$ for i = 1,2,...,k, and generates a proof of knowledge $\pi_{pok} \leftarrow \mathcal{P}(V||x||\omega_1||...||\omega_k; g; x; t)$. The ciphertext is $c = (x, \omega_1, \omega_2, ..., \omega_k, \pi_{pok})$.

- The extraction algorithm $EXT$(c) where c is a ciphertext, first verifies the proof $\pi_{pok}$, computes $\tilde{\omega} = \omega_1 \omega_2 ... \omega_k$, and outputs the naked ciphertext $\tilde{c} = (x, \tilde{\omega})$.

- The transformation algorithm $\mathcal{T}(dk_2, V, \text{s}, \text{c})$ verifies the proof $\pi_{pok}$, computes $\bar{x} = x^s, \bar{\omega_i} = \omega_i^s$ and $\hat{\omega_i} = \bar{x}^{a_{2,i}}$ for i = 1,2,...,k, generates proofs

  $$\bar{\pi} \leftarrow \mathcal{P}_{eqdl}(c, g, s; x, \omega_1, ..., \omega_k; \bar{x}, \bar{\omega_1}, ..., \bar{\omega_k}),$$
  $$\hat{\pi} \leftarrow \mathcal{P}'_{eqdl}(c, g, \bar{x}; a_{2,1}, ..., a_{2,k}; \hat{\omega_1}, ..., \hat{\omega_k})$$

  and outputs the pre-code ciphertext $\breve{c} = (\bar{x}, \bar{\omega_1}, ..., \bar{\omega_k}, \hat{\omega_1}, ..., \hat{\omega_k}, \bar{\pi}, \hat{\pi})$.

- The pre-code decryption algorithm $\mathcal{D}_R(dk_3, V, \gamma, \mathrm{c}, \breve{c})$ verifies $\pi_{pok}, \bar{\pi}, \hat{\pi}$, computes $\rho_i = \bar{\omega_i}\hat{\omega_i}\bar{x}^{-a_{3,i}}$ for i = 1,2,...,k and outputs precodes $\vec{\rho} = (\rho_1, ..., \rho_k)$.
- The decryption protocol is not specified.
- The decryption algorithm first verifies $\pi_{pok}$, then computes $m_i = \omega_i x^{-a_{1,i}}$ for i = 1,2,...,k. Note: decryption key must be $(a_{1,i}, ...)$ instead of sum.

## 5.3 Functionalities

The ideal functionality describes the functional and security requirements for the voting protocol and is parameterized by the function leak:

$$leak(\Delta, V, P, \vec{m}) = \begin{cases} \vec{m} & P \text{ corrupt,} \\ \Delta(\vec{m}) & R \text{ corrupt, and} \\ \perp & \text{otherwise.} \end{cases}$$

where $\Delta$ is a permutation of the options.

The secure channel functionality and the phone channel functionality describes assumptions about communication. Let $\mathcal{V}$ be the set of voters, $\mathcal{P}$ a set of computers and $\mathcal{F}$ a set of phones. Phones are considered personal, so denote a voters phone $F_V$. It is assumed that there are *confidential, identified and authenticated channels between the infrastructure players, between voters and computers, between voters and their own phones, and a one-way channel from the return code generator R to the voters phones.* [3]

The Phys-vote functionality describes how physical voting in the polling station happens.

The PKI functionality allows voters to delegate their identity to computers. After this delegation the computer can digitally sign for the voter and establish an authenticated connection to a ballot box. Most of the technicalities of using the PKI functionality has been ignored to increase readability in [3], but it is important to know that it is there.

The return code generator functionality describes the cryptographic infrastructure that the return code generator has a signing key, and the ballot box and computers have the corresponding key for verification.

It is assumed that all keys are generated by a trusted dealer as the setup phase was not the focus of [3].

*Setup phase*

On (keygen) from $U$:

1. Send (keygen, $U$) to $\mathcal{S}$, and wait for (keygen, $U$, b) from $\mathcal{S}$.
2. If $b = 0$, send (reject) to $U$ and stop. Otherwise, send (accept) to $U$.
3. If $R$ is corrupt, then for each voter $V$ choose a random permutation $\Delta_V$ on $O$ (options).
4. If all setup phase players have sent (accept), enter ballot submission phase.

*Ballot submission phase*

On (use, $P$) from $V$:

1. Hand over (using, $P$, $V$) to $\mathcal{S}$ and wait for (using, $P$, $V$) from $\mathcal{S}$. Record (use, $V$, $P$). Send (using) to $V$.

On (vote, $P$, $\vec{m}$) from $V$:

1. Select the next sequence number *seq*.
2. Stop if (use, $V$, $P$) is not recorded, or if $V$ is marked as voting. Mark $V$ as voting.
3. Hand over $(voting, seq, V, P, leak(\Delta_V, V, P, \vec{m}))$ to $\mathcal{S}$. Wait for (store, $V$, $b_1$), (notify, $V$, $b_2$) and (finish, $V$, $b_3$) from $\mathcal{S}$.
   - If $P$ is honest: If $b_1 = 1$, erase any record (ballot, $V$, ...), record (ballot, $V$, $\vec{m}$) and append (notify, $\vec{m}$) to $V$'s queue. Otherwise, record (lost, *seq*, $V$, $\vec{m}$).
   - If $b_2 = 1$, pop the first entry (notify, $\vec{m}'$) from $V$'s queue.

   Between messages, hand over (done) to $\mathcal{S}$.
4. If $b_1 = b_2 = b_3 = 1$ and $\vec{m}' \sim \vec{m}$, send (accept) to $V$, otherwise send (reject) to $V$.
5. Mark $V$ as not voting.

Figure 5.1a: The Ideal Functionality, parameterized by the leak function

*Ballot submission phase (continued)*

On (replay, *seq*) from $\mathcal{S}$:

1. Stop unless (lost, *seq*, $V$, $\vec{m}$) is recorded.
2. Erase the record (lost, *seq*, $V$, $\vec{m}$) and any record (ballot, $V$, ...).
3. Record (ballot, $V$, $\vec{m}$). Append (notify, $\vec{m}$) to $V$'s queue.
4. Hand over (forged) to $\mathcal{S}$.

On (forge, $V$, $P$, $\vec{m}$) from $\mathcal{S}$:

1. Ignore unless $V$ and $P$ are corrupt, or (use, $V$, $P$) is recorded.
2. Erase any record (ballot, $V$, ...). Record (ballot, $V$, $\vec{m}$).
3. If $V$ is honest, append (notify, $\vec{m}$) to $V$'s queue.
4. Hand over (forged) to $\mathcal{S}$.

On (notify, $V$, b) from $\mathcal{S}$:

1. If $b = 1$, discard the first entry from $V$'s queue.
2. Send (forgery) to $V$.

On (close) from $B$, or from $\mathcal{S}$ if $B$ is corrupt:

1. Enter counting phase. Any (vote, ...) messages currently being processed will proceed as above, subject to the requirement that the first step (step, ..., $b_1$) must have $b_1 = 0$.
2. Let (ballot, $V_1, \vec{m}_1$), ..., ($ballot, V_n, \vec{m}_n$) be all the ballot records, sorted by ballot. Hand over (closing, $\vec{m}_1, ..., \vec{m}_n$) to $\mathcal{S}$.

Figure 5.1b: The Ideal Functionality, parameterized by the leak function

*Counting phase*

On (voter, $P$, $\vec{m}$) from $V$:

1. Select the next sequence number $seq$.
2. Stop voter is marked as voting. Mark $V$ as voting.
3. Hand over $(voting, seq, V, P, leak(\Delta_V, V, P, \vec{m}))$ to $\mathcal{S}$. Wait for (store, $V$, $b_1$), (notify, $V$, $b_2$) and (finish, $V$, $b_3$) from $\mathcal{S}$.

   Between messages, hand over (done) to $\mathcal{S}$.
4. Send (reject) to $V$.
5. Mark $V$ as not voting.

On (finish, $D$, b) from $\mathcal{S}$:

1. If $b = 0$ and either $B$, $R$ or $A$ is corrupt, send (reject) to $D$. Otherwise, let (ballot, $V_1, \vec{m}_1), ..., (ballot, V_n, \vec{m}_n)$ be all the ballot records, send (result, sort$(\omega(\vec{m}_1), ..., \omega(\vec{m}_n))$) to $D$.

On (finish, $A$, b) from $\mathcal{S}$:

1. If $b = 0$ and either $B$, $R$ or $D$ is corrupt, send (reject) to $A$. Otherwise, let (ballot, $V_1, \vec{m}_1), ..., (ballot, V_n, \vec{m}_n)$ be all the ballot records, send (result, sort$(\omega(\vec{m}_1), ..., \omega(\vec{m}_n))$) to $A$.

Figure 5.1c: The Ideal Functionality, parameterized by the leak function

On (send, $\mathcal{X}$, m) from $\mathcal{Y}$:

1. If $(\mathcal{X}, \mathcal{Y})$ is in $\mathcal{V} \times \mathcal{P}$, $\mathcal{P} \times \mathcal{V}$, $\{A, B, D, R\}^2$, or $\{(V, F_V) \mid V \in \mathcal{V}\}$, then send (recv, $\mathcal{Y}$, m) to $\mathcal{X}$.

Figure 5.2: Secure channel functionality

On (send, $F$, m) from $R$:

1. Push m onto $F$'s queue. Hand over (sending, $F$) to $\mathcal{S}$

On (deliver, $F$) from $\mathcal{S}$:

1. Pop m from the front of $F$'s queue. Send (recv, $R$, m) to $F$.

Figure 5.3: Phone channel functionality

On (vote, $\vec{m}$) from $V$:

1. If ($V$, $\vec{m}$) is registered, stop.
2. Register ($V$, $\vec{m}$).

On (voters) from $B$ or $A$:

1. Output a sorted list ($\{V \mid (V, \vec{m}) \text{ is recorded}\}$).

On (votes) from $A$ or $D$:

1. Output a sorted list ($\{\vec{m} \mid (V, \vec{m}) \text{ is recorded}\}$)

Notes on the physical vote tallying that is not specified in the protocol:

1. The ballot box gathers a list of voters who have voted physically, and then erases the electronic votes registered on these voters.
2. The decryptor collects the physical votes and tallies them.
3. The auditor does both to verify that it was done correctly.

Figure 5.4: Phys-vote functionality

On (use, $P$) from $V$:

1. Hand (using, $V$, $P$) to $\mathcal{S}$, and wait for (using, $V$, $P$). Record (use, $V$, $P$). Send (using) to $V$.

On (establish, $B$, $V$) from $P$:

1. Stop unless (use, $V$, $P$) is recorded.
2. Choose a unique session identifier *sid*. Hand over (establish, *sid*, $B$, $V$, $P$) to $\mathcal{S}$
3. Wait for (established, *sid*) from $\mathcal{S}$. Record (session, *sid*, $V$, $P$, 0). Send (established, *sid*, $P$, $V$) to $B$ and hand over (establish, *sid*, $B$, $V$, $P$) to $\mathcal{S}$.
4. Wait for (established, *sid*) from $\mathcal{S}$. Record (session, *sid*, $V$, $P$, 1). Send (established, *sid*, $B$, $V$) to $P$

On (send, *sid*, m) from $U$:

1. If $U = B$, set $b = 0$, otherwise set $b = 1$.
2. Stop unless (session, *sid*, ..., b) is recorded.
3. Push m into the *sid*-b queue and hand over (sending, *sid*, $|m|$, b) to $\mathcal{S}$

On (deliver, *sid*, b) from $\mathcal{S}$

1. Stop unless (session, *sid*, ..., $P$, 1) is recorded.
2. Pop m off the front of the *sid*-b queue. If $b = 0$, send (recv, *sid*, m) to $P$, otherwise send (recv, *sid*, m) to $B$.

On (sign, $V$, m) from $P$:

1. Stop unless (use, $V$, $P$) recorded.
2. If no record (keys, $V$, *sk*, *vk*) exists, compute (*sk*, *vk*) $\leftarrow \mathcal{K}_s$ and record (keys, $V$, *sk*, *vk*).
3. Choose a unique signing identifier *sid*. Hand over (signing, *sid*, $V$, $P$) to $\mathcal{S}$ and wait for (signing, *sid*, $V$, $P$).
4. Compute $\sigma \leftarrow \mathcal{S}_s(sk, m)$. Record (signature, $V$, m, $\sigma$, 1). Send (signature, $V$, m, $\sigma$) to $P$.

On (verify, $V$, m, $\sigma$) from $U$:

1. If no record (keys, $V$, *sk*, *vk*) exists, or (signature, $V$, m, $\sigma$, 1) is not recorded, record (signature, $V$, m, $\sigma$, 0).
2. If (signature, $V$, m, $\sigma$, b) is recorded, send (verified, $V$, m, $\sigma$, b) to $U$.

Figure 5.5: PKI functionality, parameterized over a signature scheme $(\mathcal{K}_s, \mathcal{S}_s, \mathcal{V}_s)$.

On (sign,m) from $R$:

1. If no record (keys, $R$, $sk$, $vk$) exists, compute ($sk$, $vk$) $\leftarrow \mathcal{K}_s$ and record (keys, $R$, $sk$, $vk$).
2. Compute $\sigma \leftarrow \mathcal{S}_s(sk, \text{m})$. Record (signature, $R$, m, $\sigma$, 1). Send (signature, m, $\sigma$) to $R$.

On (verify, $R$, m, $\sigma$) from $U$:

1. If no record (keys, $R$, $sk$, $vk$) exists, or (signature, $R$, m, $\sigma$, 1) is not recorded, record (signature, $R$, m, $\sigma$, 0).
2. If (signature, $R$, m, $\sigma$, b) is recorded, send (verified, $R$, m, $\sigma$, b) to $U$

Figure 5.6: Return code generator functionality, parameterized over a signature scheme $(\mathcal{K}_s, \mathcal{S}_s, \mathcal{V}_s)$.

On (keygen) from $U$

1. If this is the first (keygen) message received, do:
   - ($ek$, $dk_1$, $dk_2$, $dk_3$) $\leftarrow \mathcal{K}$
   - For each voter $V$, compute $(s_V, \gamma_V) \leftarrow \mathcal{L}(ek, V)$ and sample a random injective function $D_V$ from the image of $s_V$ to $C_H$.
2. Hand over (keygen, $U$) to $\mathcal{S}$

On (keygen, $U$, b) from $\mathcal{S}$:

1. If $b = 1$, send (keys, material) to $U$, where material is as detailed in Table 1 of [3]. Otherwise send (reject) to $U$.
2. If (keygen, $U$, 1) has been received for every infrastructure player and every elecoral board player, then for each voter $V$, send (keys, material) to $V$, where material is detailed in Table 1 of [3].

Figure 5.7: Trusted dealer for the setup phase

## 5.4 Protocol

The voters protocol describes how a voter should vote, and how to deal with coercion.

The phones protocol is simply to send all incoming messages to the voter.

The computers protocol encrypts the vote, sign the encrypted vote and sends the encrypted vote as well as the signature to the ballot box. When getting a response the computer decides whether to accept or not.

The ballot box protocol describes how it works together with the return code generator as well as tallying details.

The return code generators protocol describes how it works together with the ballot box to generate return codes.

The decryptors protocol decrypts the set of encrypted votes and outputs the election result.

The auditors protocol deals with verification of the execution of protocols.

*Setup phase*

On (keys, T) from key generation functionality:

1. Remember the set T.

*Ballot submission phase*

On (coerce, msg) from Phys-com:

1. Remember that you are coerced.
2. Interpret $\vec{m} \leftarrow$ msg.
3. Proceed as if input (vote, $P$, $\vec{m}$) was recieved from $\mathcal{Z}$.

On (release, msg) from Phys-com:

1. "Forget" that you are coerced.
2. If compliant with coercion stop here.
3. If $\vec{m}$ is remembered do the steps below, otherwise wait until $\vec{m}$ is interpreted from $\mathcal{Z}$ and then do the steps below:
   (a) Stop if $V$ has voted physically previously.
   (b) Stop if already voting. Mark $V$ as voting.
   (c) Send (vote, $\vec{m}$) to Phys-vote.
   (d) Mark $V$ as not voting.
4. Alternatively: If $\vec{m}$ is remembered, proceed as if input (vote, $P$, $\vec{m}$) was recieved from $\mathcal{Z}$. Otherwise proceed as if input (vote, $P$, $\vec{0}$) was recieved from $\mathcal{Z}$.

Figure 5.8a: The Voter

*Ballot submission phase (continued)*

On (use, $P$) from $\mathcal{Z}$:

1. Send (use, $P$) to PKI func., wait for (using) in return and record (use, $V$, $P$).
2. If $V$ is coerced, send (use, $V$, $P$) to $\mathcal{A}$. *(It is weird that the user have to send this to the adversary, but no solution has made itself apparent)*

On (vote, $P$, $\vec{m}$) from $\mathcal{Z}$:

1. Stop if already voting or (use, $V$, $P$) is not recorded. Mark as voting. Remember $\vec{m}$ if not coerced.
2. Send (vote, $\vec{m}$) to $P$.
3. Wait for (accept) from $P$ and (codes, $\vec{r}$) from $F$, or (reject) from $P$.
4. If (reject) was received from $P$, output (reject) and stop.
5. Let $\vec{m} = (m_1, m_2, ..., m_k), \vec{r} = (r_1, r_2, ..., r_k)$. If $(m_i, r_i) \notin T$ for some i, output (reject). Otherwise output (accept). Mark as not voting.

On (codes, ...) from $F$:

1. Output (forgery).

Figure 5.8b: The Voter

On m from $R$:

1. Send m to $V$

Figure 5.9: The Phone

*Setup submission phase*

On (keys, $ek$) from key generation functionality:

1. Remember $ek$.

*Ballot submission phase*

On (vote, $\vec{m}$) from $V$:

1. If any step below fails, send (reject) to $V$ and stop.
2. Establish a connection to the ballot box $B$.
3. $c \leftarrow \mathcal{E}(ek, V, \vec{m})$
4. Sign $c$ on behalf of $V$ to get a signature $\sigma$.
5. Send ($V$, c, $\sigma$) (using a fixed-length encoding of ($V$, c, $\sigma$)) to $B$ and wait for $\sigma_R$ from $B$.
6. Verify that $\sigma_R$ is $R$'s signature on $H(V, c)$, then send (accept) to $V$.

Figure 5.10: The Computer

*Setup submission phase*

On input (keygen):

1. Send (keygen) to the key generation functionality.
2. Wait for (keys, $dk_2$, $\{(V, s_V)\}$) or (reject) from the key generation functionality.
3. In the former case, remember $dk_2$and $\{(V, s_V)\}$, and output (accept). Otherwise output (reject).

*Ballot submission phase*

On (established, *sid*, $P$, $V$) from PKI functionality:

1. If any step below fails, send (reject) to $P$ and stop.
2. Wait for ($V$, c, $\sigma$) from $P$.
3. Verify that $\sigma$ is $V$'s signature on c.
4. If the voter $V$ is marked as voting, wait until $V$ is no longer marked as voting. Mark $V$ as voting.
5. Select the next sequence number *seq*
6. Compute $\breve{c} = \mathcal{T}(dk_2, V, s_V, c)$, and verify that $\breve{c} \neq \perp$.
7. Send (ballot, *seq*, $V$, $c, \sigma, \breve{c}$) to $R$ and wait for (receipt, $seq, \sigma_R$) from $R$.
8. Verify that $\sigma_R$ is $R$'s signature on $H(V$, c).
9. Record (ballot, *seq*, $V$, c, $\sigma$), mark $V$ as not voting and send $\sigma_R$ to $P$.

Figure 5.11a: The Ballot Box

*Voting phase (continued)*

On input (close):

1. Enter counting phase. Wait until every ballot submission currently in progress has finished.
2. Send (close) to $R$.
3. Let $L_1$ be the list of all ballot records. For each voter, select the ballot record with maximal sequence number, resulting in the records $(ballot, seq_1, V_1, c_1, \sigma_1)$, ..., $(ballot, seq_n, V_n, c_n, \sigma_n)$
4. Compute $\tilde{c_i} = EXT(c_i)$, $i = 1, 2, ..., n$.
5. Let $L_2$ be the sorted list of naked ciphertexts $\tilde{c_1}, \tilde{c_2}, ..., \tilde{c_n}$
6. Send (decrypt, $L_2$) to $D$ and (content, $L_1$) to $A$.

*Counting phase*

On (established, *sid*, $P$, $V$) from PKI functionality:

1. Ignore.

Figure 5.11b: The Ballot Box

*Setup phase*

On input (keygen):

1. Send (keygen) to the key generation functionality.
2. Wait for (keys, $dk_3$, $\{(V, \gamma_V, D_V)\}$) or reject from the key generation functionality.
3. In the former case, remember $dk_3$and $\{(V, \gamma_V, D_V)\}$, and output (accept). Otherwise output (reject).

*Ballot submission phase*

On (ballot, *seq*, $V$, $c, \sigma, \check{c}$) from $B$:

1. If any step below fails, send (reject, seq) to $B$ and stop.
2. Verify that $V$ is not marked as voting. Mark $V$ as voting.
3. Compute $h \leftarrow H(V, \text{c})$. Verify that no records (ballot, $V$, ..., h) or (ballot, $V$, $seq'$), $seq \leq seq'$, exist. Record (ballot, $V$, seq, h).
4. Verify that $\sigma$ is $V$'s signature on c.
5. Compute $\vec{\rho} \leftarrow D_R(dk_3, V, \gamma_V, c, \check{c})$, and verify that $\vec{\rho} \neq \perp$ and that $\rho_i$ is in the domain of $D_V$ for $i = 1, 2, ..., k$
6. Compute $r_i = D_V(\rho_i)$ for $i = 1, 2, ..., k$.
7. Sign $h$ to get the signature $\sigma_R$.
8. Send $\vec{r}$ to $F_V$ and (receipt, seq, $\sigma_R$) to $B$. Mark $V$ as not voting.

On (close) from $B$:

1. Verify that all ballot processing is done.
2. Let $L_3$ be the list of all ballot records. Send (hashes, $L_3$) to $A$.

*Counting phase*

On (ballot, *seq*, $V$, $c, \sigma, \check{c}$) from $B$:

1. Ignore.

Figure 5.12: The Return Code Generator

*Setup phase*

On input (keygen):

1. Send (keygen) to the key generation functionality.
2. Wait for (keys, $dk_1$) or (reject) from the key generation functionality.
3. In the former case, remember $dk_1$and output (accept). Otherwise output (reject).

*Counting phase*

On (decrypt, $L_2$) from $B$:

1. Send (hash, H($L_2$)) to $A$. Wait for (accept) from $A$.
2. Run the prover role of the decryption protocol $\Pi_{DP}$ with $dk_1$ as private input and $L_2$ as public input, and with $A$ playing the verifier role.
3. If the decryption protocol output is $\perp$, output (reject).
4. If the decryption protocol output is $\vec{m_1}, ..., \vec{m_n}$ then output (result, sort($\vec{m_1}, ..., \vec{m_n}$)).

Figure 5.13: The Decryptor

*Setup phase*

On input (keygen):

1. Send (keygen) to the key generation functionality.
2. Wait for (keys, $ek$) or (reject) from the key generation functionality.
3. In the former case, remember $ek$ and output (accept). Otherwise output (reject).

*Counting phase*

On (content, $L_1$) from $B$:

1. Wait for (hashes, $L_3$) from $R$ and (hash, h) from $D$.
2. Verify that for each ballot (ballot, seq, $V$, c, $\sigma$) in $L_1$, $\sigma$ is $V$'s signature on c.
3. Verify that for each ballot (ballot, seq, $V$, c, $\sigma$) in $L_1$, there is a corresponding record (ballot, $V$, seq, $h'$) in $L_3$ with $h' = H(V, \text{c})$ and vice versa.
4. For each voter, select the ballot record with maximal sequence number from $L_1$, resulting in records (ballot, $seq_1, V_1, c_1, \sigma_1$),...,(ballot,$seq_n, V_n, c_n, \sigma_n$)
5. Compute $\tilde{c} = EXT(c_i), i = 1, 2, ..., n$
6. Verify that $h$ equals $H(sort(\tilde{c_1}, ..., \tilde{c_n}))$.
7. Send (accept) to $D$.
8. Run the verifier role of the decryption protocol $\Pi_{DP}$ with $ek$and $L_2$ as public input, and with $D$ playing the prover role.
9. If the decryption protocol output is $\perp$, output (reject).
10. If the decryption protocol output is $\vec{m_1}, ..., \vec{m_n}$, then output (result, sort($\vec{m_1}, ..., \vec{m_n}$))

Figure 5.14: The Auditor

## 5.5 Analysis

In the Norwegian Voting Protocol $M_I$ is empty, while $M_E$ holds a list T of precomputed option-return code pairs printed on the voters voting card. Hence, if a voter is coerced he does not have any secrets. Cast ballots are kept secret in the voting system, as well as whether a voter has participated or not. The adversary could erase $M_E$, trying to stop the voter from voting, and in this case the voter should be able to identify himself at a polling station and vote physically.

Should the voter submit his vote wrongly he may revote to fix his mistakes. Not all voters will bother with checking codes, but many will and if something is wrong with the voters codecards and the codes recieved on the phone it is likely that it will be discovered. Should the voter loose his codes he may identify himself at a physical polling booth and vote there.

The coercion-strategy is to follow instructions from the adversary. Whatever vote the adversary wants the coerced voter to cast, the cast vote can be trumped by revoting. The revoting takes place in the recovery-strategy, where the voter either revotes electronically or votes physically.

It is possible that the adversary can coerce a voter until the ballot box closes, so that the voter is unable to revote. This can not be fixed, and it is a valid attack against the voting system. The Norwegian Voting Protocol breaks down in several ways against a mixed adversary, but that is not the focus of this master thesis. Studying the protocol with regards to a mixed adversary is an interesting direction for future work.

### 5.5.1 Weak coercion-resistance

When a voter is coerced by a weak adversary the voter can simply revote electronically and thus make his previous electronic vote given under coercion not count. The adversary can not see the cast ballots of voters (except the ones cast under coercion), and can not know if the ballots cast under coercion have been trumped by a revote. Demanding that a voter abstain from voting, the only way to gain evidence that the voter is non-compliant is if the voter votes during the coercion-stage. However, in the coercion-stage the voter follows the adversary's instructions and will not vote. Hence the Norwegian Voting Protocol is weak coercion-resistant.

### 5.5.2 Strong coercion-resistance

Against a strong adversary the voter can vote physically which trumps electronic votes, a recovery-strategy that does have some problems. When a

voter is released he might not have decided what to vote. With electronic revoting the voter could submit a blank vote to "erase" the adversary's vote, but with physical voting a blank vote trumps all other votes that come after. Hence the voter can not cast his vote if he later decides how he wants to vote. Therefore the voter should have some time to decide what to vote after being released. In the voting system electronic voting is allowed until election day, a day in which only physical voting is allowed. Election day is a natural time for an adversary to release coerced voters as they no longer can vote on their devices. The adversary might coerce longer if he has resources to monitor that voters do not go to a physical voting station. Voters who has not voted physically after being released from coercion can use election day to do so, and if they have not decided what to vote as the election is about to close they must submit a blank ballot.

Assume that the adversary releases the voter before the physical polling stations and the ballot box closes. As cast ballots are kept secret, the adversary can not determine whether a ballot submitted from a coerced voter will be tallied as it may have been trumped by a physical vote. Coercing voters into abstaining will not work as the adversary does not know which voters have voted physically. Hence the adversary can not know whether coercion succeeded, and the Norwegian Voting Protocol is strong coercion-resistant.

# Chapter 6

# Conclusion

## 6.1 Prêt à Voter

There are viable attacks against Prêt à Voter, and two well known attacks are the randomization attack and the chain voting attack. By leaving the optional part of the polling station functionality out an ideal case was built in such a way that any property of the ideal case is also a property of the real case. Then the following conclusion was made:

**Conclusion.** *Prêt à Voter described in the real case where the optional part of $F$ is not included is not resistant against randomization attacks. If the attack is not a randomization attack a voter can vote for his intended candidate, and if no such candidate exists the voter can vote for a random candidate.*

## 6.2 Civitas

With both credentials in $M_E$ Civitas is not safe from a strong adversary. The adversary can instruct a coerced voter to vote with both credentials, and if the voter revotes after being released the adversary sees it. Against a weak adversary the voter can revote without being detected. A mixed adversary with access to registration tellers will be able to decide which credential is real and which is fake for some voters, and thus breaking the security of the system. Hence we get:

**Conclusion.** *When both credentials are stored externally Civitas is weak coercion-resistant if revotes are allowed, but not strong coercion-resistant.*

With both credentials in $M_I$ it is possible to submit fake credentials and fake votes, so against a weak adversary a voter will win. A strong adversary

will detect any real votes cast by a released voter though, so Civitas is not safe against a strong adversary. Mixed adversaries with access to registration tellers may be able to decide whether credentials are real or fake, and may also detect a voter handing over two fake credentials. There are several problems with having both credentials in $M_I$, and it does not seem like a good idea unless credentials are of very small size.

**Conclusion.** *When both credentials are stored internally Civitas is weak coercion-resistant, but not strong coercion-resistant.*

## 6.3 The Norwegian Voting Protocol

The Norwegian Voting Protocol is safe against a weak adversary. Voters follow the adversary's instructions while coerced, and when released revote electronically for their preferred candidate or submit a blank vote. Against a strong adversary the protocol is still safe, but now voters must vote physically after being released from coercion. Mixed adversaries may cause trouble, but that is not in the focus of this master thesis.

**Conclusion.** *Assuming that voters are released from coercion before the polling stations and ballot boxes close, the Norwegian Voting Protocol is both weak and strong coercion-resistant.*

## 6.4 Future work

Prêt à Voter is the voting system described with most detail in this master thesis. Assumptions are made about the scanner and the election authority being honest, and removing these assumptions could lead to a useful analysis. Giving the adversary more power by allowing the interception and editing of messages could also lead to some interesting results.

Civitas and the Norwegian Voting Protocol are discussed in a not very detailed manner. Describing these systems more detailed with care for implementation and then doing a more complete analysis is something that would certainly be of use. One could also try to alter the Norwegian Voting Protocol to get end-to-end verifiability without loosing the coercion-resistance of the system, or at least look at what one would need to do to get end-to-end verifiability and if this is possible.

# Bibliography


[1] David Chaum, Peter Y.A. Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In Sabrinade Capitani Vimercati, Paul Syverson, and Dieter Gollmann, editors, *Computer Security-ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer Berlin Heidelberg, 2005.

[2] Kristian Gjøsteen. Is end-to-end verifiable remote voting possible in Norway? `http://uni.lu/snt/research/apsia/events/verifiable_voting_schemes_workshop_from_theory_to_practice`, 2013. [Accessed online 29-October-2013].

[3] Kristian Gjøsteen. The norwegian internet voting protocol. Cryptology ePrint Archive, Report 2013/473, 2013. `http://eprint.iacr.org/`.

[4] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y.A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 37–63. Springer Berlin Heidelberg, 2010.

[5] Microsoft. Microsoft Security Intelligence Report Volume 14. `http://www.microsoft.com/security/sir/archive/default.aspx`, 2012. [Accessed online 27-May-2013].

[6] Andrew C. Myers, Michael Clarkson, and Stephen Chong. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368. IEEE, May 2008.

[7] P. Y. A. Ryan. Prêt à voter with paillier encryption, 2006.

[8] P Y A Ryan and S A Schneider. Prêt à voter; voter with re-encryption mixes. In *Proceedings of the 11th European Conference on Research*

*in Computer Security*, ESORICS'06, pages 313–326, Berlin, Heidelberg, 2006. Springer-Verlag.

[9] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à voter: A voter-verifiable voting system. *IEEE Transactions on Information Forensics and Security*, 4(4):662–673, December 2009.